DINOS: Data INspired Oligo Synthesis for DNA Data Storage

KEVIN VOLKEL, KYLE J. TOMEK, ALBERT J. KEUNG, and JAMES M. TUCK,
North Carolina State University

As interest in DNA-based information storage grows, the costs of synthesis have been identified as a key bottleneck. A potential direction is to tune synthesis for data. Data strands tend to be composed of a small set of recurring code word sequences, and they contain longer sequences of repeated data. To exploit these properties, we propose a new framework called DINOS. DINOS consists of three key parts: (i) The first is a hierarchical strand assembly algorithm, inspired by gene assembly techniques that can assemble arbitrary data strands from a small set of primitive blocks. (ii) The assembly algorithm relies on our novel formulation for how to construct primitive blocks, spanning a variety of useful configurations from a set of code words and overhangs. Each primitive block is a code word flanked by a pair of overhangs that are created by a cyclic pairing process that keeps the number of primitive blocks small. Using these primitive blocks, any data strand of arbitrary length can be assembled, theoretically. We show a minimal system for a binary code with as few as six primitive blocks, and we generalize our processes to support an arbitrary set of overhangs and code words. (iii) We exploit our hierarchical assembly approach to identify redundant sequences and coalesce the reactions that create them to make assembly more efficient.

We evaluate DINOS and describe its key characteristics. For example, the number of reactions needed to make a strand can be reduced by increasing the number of overhangs or the number of code words, but increasing the number of overhangs offers a small advantage over increasing code words while requiring substantially fewer primitive blocks. However, density is improved more by increasing the number of code words. We also find that a simple redundancy coalescing technique is able to reduce reactions by 90.6% and 41.2% on average for decompressed and compressed data, respectively, even when the smallest data fragments being assembled are 16 bits. With a simple padding heuristic that finds even more redundancy, we can further decrease reactions for the same operating point up to 91.1% and 59% for decompressed and compressed data, respectively, on average. Our approach offers greater density by up to 80% over a prior general purpose gene assembly technique. Finally, in an analysis of synthesis costs in which we make 1 GB volume using *de novo* synthesis versus making only the primitive blocks with *de novo* synthesis and otherwise assembling using DINOS, we estimate DINOS as $10^5\times$ cheaper than *de novo* synthesis.

CCS Concepts: • **Mathematics of computing → Permutations and combinations**; • **Information systems → Information storage technologies**; • **Hardware → Emerging architectures;**

Additional Key Words and Phrases: DNA information storage, DNA synthesis, DNA assembly algorithms

This work was supported with funding by the National Science Foundation (Grants CNS-1650148, CNS-1901324, and ECCS 2027655), a North Carolina State University Research and Innovation Seed Funding Award (Grant 1402-2018-2509), and a Department of Education Graduate Assistance in Areas of Need Fellowship.

Authors' addresses: K. Volkel, 603 Austin Ave. Cary, NC 27511; email: kvolkel@ncsu.edu; K. J. Tomek, 4101 The Oaks Dr. Raleigh, NC 27606; email: kjtomek@ncsu.edu; A. J. Keung, North Carolina State University, Campus Box 7905, Raleigh, NC 27695-7905; email: ajkeung@ncsu.edu; J. M. Tuck, NC State University, Dept. of ECE PO Box 7911 Raleigh, NC 27695-7911; email: jtuck@ncsu.edu.

1550-4832/2022/06-ART53
https://doi.org/10.1145/3510853

ACM Reference format:
Kevin Volkel, Kyle J. Tomek, Albert J. Keung, and James M. Tuck. 2022. DINOS: Data INspired Oligo Synthesis for DNA Data Storage. *J. Emerg. Technol. Comput. Syst.* 18, 3, Article 53 (June 2022), 35 pages.
https://doi.org/10.1145/3510853

1 INTRODUCTION

The exponential growth of data in the world is expected to reach 175 ZB by 2025 [32]. To meet this growing demand, DNA has emerged as a viable medium for high-density storage of archival information, with lifetimes over 100,000 years [17] and storage densities seven orders of magnitude larger than magnetic tape [11]. However, a major impediment to the adoption of DNA is the seven orders of magnitude higher cost of DNA synthesis as compared to DNA sequencing [9]. Hence, large breakthroughs in synthesis technology that substantially lower costs are paramount. The high costs of synthesis stem, in part, from the goal of producing arbitrary genetic sequences, where the sequence has a specific biological meaning. The production of such sequences needs high fidelity so it can be integrated into biological systems and achieve the desired result. However, DNA-based storage systems are not similarly constrained, opening the door to data-centric synthesis approaches.

Data INspired Oligo Synthesis (DINOS) builds on top of two main observations to form a data-driven approach to synthesis. The first observation is that DNA-based storage systems have the freedom to select arbitrary sequences to represent information. This provides the flexibility to allow sequence design to aid the process of synthesis. With the observation that DNA is capable of self-assembling thanks to Watson-Crick pairings among single strands of DNA, it has been shown that long linear assemblies can be constructed from a set of DNA parts [5, 6, 22, 28, 36–38, 43, 44, 46]. Thus, the customization of strand design for data storage can be leveraged so strands self-assemble messages of information. Although DNA assembly is well studied, little if any work has been done on leveraging DNA assembly specifically for creating data strands.

The second observation is that data contains a substantial amount of redundancy and repetition. There is one level of repetition formed by the repeated use of the same symbols at the bit or byte granularity. For example, only two DNA sequences are needed to represent a binary code, and these two sequences can be assembled to form arbitrary binary messages. Such an approach has the potential of decreasing synthesis costs greatly due to the fact that creating many copies of a small set of DNA sequences can be considerably cheaper than synthesizing many unique instances as is done in current synthesis techniques [1, 2]. Such an approach is also being explored by emerging DNA information storage companies such as Catalog Technologies Inc. [34].

Data redundancy also occurs at larger granularity. Sequences of bits are often repeated both within and across strands and files. Rather than separately and repeatedly assembling the same sequences of bits, we should consolidate redundant assemblies to streamline and reduce the number of reactions and time required. With an appropriate representation of strand assembly, we can identify redundant assemblies and consolidate them into a single reaction, in much the same way that compiler algorithms identify redundant expressions and eliminate them [21].

DINOS consists of three key parts that are co-designed to work together and offer several tunable features that allow it to be optimized for lower or higher information density, longer or shorter strands, or reduced assembly reactions. First, DINOS is built on top of a set of primitive DNA blocks with properties that enable the assembly of arbitrary data strands. These primitive blocks consist of two parts: a code word sequence and a special overhang sequence on each end. Each symbol of information is represented by a code word, and assembly of strands is enabled by the overhang regions. A unique aspect of our approach is how we combine code words and overhangs. Rather

than requiring all possible combinations, we only require $|O| \cdot 2^k$ primitive blocks, where $|O|$ is the number of overhangs and 2^k is the number of unique code words.[1] Our approach allows us to describe a range of systems that can be constructed with varying numbers of primitive blocks by independently choosing the number of code words or overhangs. We study a range of choices for k and $|O|$ and explain their merits and tradeoffs.

The second part of DINOS is the hierarchical assembly algorithm. The hierarchical assembly algorithm is conceived of as an n-ary tree data structure. Primitive blocks are arranged at the leaves of the tree, and the tree's nodes represent reactions. Internal nodes represent reactions that produce sub-strands using the results of children, and the root reaction provides the final assembly of the target DNA strand. The assembly algorithm relies on Watson-Crick pairings between primitive blocks in a reaction, and hence our algorithm ensures that primitive blocks are chosen in such a way to encode the desired data and assemble in the proper order. The depth and degree of the tree is determined by the number of overhangs, which is one of our tunable design parameters. This allows for tradeoffs between tall binary trees with relatively more reactions versus shorter $|(O-1)|$-ary trees with fewer reactions. Our assembly algorithm when paired together with our primitive block design leads to storage densities up to 80% better than a state-of-the-art gene assembly approach.

The third part of DINOS is the detection and elimination of reactions that produce redundant data [21, 45, 47]. The tree representation for assembly makes it possible to identify nodes of the tree that produce the same information. Rather than producing them many times, we can coalesce reactions to make assembly quicker and costeffective. We perform both an ideal study and a practical implementation for exploiting this redundancy. We show that a simple search over all subtrees that produce the same data along with an alignment algorithm to ensure they have matching overhangs can be effective in reducing the number of reactions performed across a corpus of information. When evaluating our redundancy elimination approaches, we show that searching for all redundancy can reduce reactions by 91% and 41.2% for decompressed and compressed data, respectively, when using a system with 16 overhangs and 1-bit code words. We show that an alignment technique can be used to further increase these reductions to 91.1% and 59%, for decompressed files and compressed files, respectively, for the same conditions. During our analysis of redundant reactions on compressed and uncompressed data, we have found that there are records in which synthesizing the decompressed version can reduce reactions by 60% compared to its compressed version. This motivates the need for further work to understand how the inherent entropy of information should be leveraged during the synthesis process.

We also studied the impact of the composition of the primitive block set from the three perspectives of physical space, storage density, and cost. We find that larger overhang sets more effectively decrease reaction count by performing more assemblies at once, while a larger number of code word symbols primarily increases storage density. However, cost of synthesis is largely affected by both parameters, and in fact requires a careful balance of both to minimize the cost associated with some fixed primitive set size. Our cost analysis also shows that with current costs of *de novo* synthesis, a ligation implemented assembly approach that leverages the ability to use primitive blocks many times after they are created with *de novo* synthesis can be five orders of magnitude cheaper than using *de novo* synthesis for an entire set of data, even if that dataset is only 1 GB and the primitive block set is the minimum of six blocks.

We organize the article as follows: Section 2 describes how data is typically encoded in DNA and the model of assembly we assume. Section 3 describes the key parts of DINOS in detail as well as

[1]We parameterize this term using 2^k as a convenience for mapping back to k-bits of binary storage, but our technique is not restricted to a power of two.

proofs of correctness. Next, Section 4 describes how to exploit redundancy in data during assembly to further reduce reactions, and Section 5 evaluates DINOS over a number of configurations. We then discuss practical considerations in Section 6 when implementing DINOS in a wet lab. Finally, we frame our work in the context of existing DNA assembly work and redundancy reduction techniques in Section 7, and Section 8 concludes.

2 BACKGROUND

2.1 Encoding a File as a Collection of DNA Strands

Information in conventional electronic storage media is usually stored as ordered sequence of binary digits, namely, $\{0, 1\}$[2]. We define the bits for a file F to be the ordered set $F_B = \{b_0, b_1, \ldots b_{z-1}\}$ where z is the length of the record in bits. Due to the high error rates constructing long strands using current synthesis technologies, we cannot make a single DNA strand long enough to hold a large file. F_B is divided into chunks of length n' bits that are feasible to synthesize. The length n' can be chosen arbitrarily, even if n' does not divide z evenly, and the last strand can be padded with extra bits if fixed length payloads are needed. Now file F is a set of chunks $F_C = \{C_0, C_1, \ldots C_{\lceil z/n' \rceil - 2}, C_{\lceil z/n' \rceil - 1}\}$. Each chunk will eventually become a DNA strand.

An important consideration in the design of these chunks is the method by which chunks are reassembled into a complete file after sequencing and decoding, since the process of synthesis and sequencing does not preserve order. There are a variety of approaches that are possible. The most prevalent and densest approach adds additional bits to each strand to represent the index of the chunk within the file [19]. The index of each strand, i, represents its order in the file, and it can be represented with a fixed number of bits, $\{a_0, \ldots, a_{log_2(\lceil z/n \rceil)-1}\}$. Hence, all chunks $C_i \in F_C$ can be represented as follows: $C_i = \{a_0, \ldots, a_{log_2(\lceil z/n' \rceil)-1}, b_{i \cdot n'}, \ldots b_{n' \cdot (i+1)-1}\}$. Note, the placement of the index vector, a, can be anywhere in the strand as long as the decoder knows where to find it during decoding. It is trivial to re-assemble the correct ordering of all bits in F_B by placing all chunks in order according to their index.

It is also common to add error correction bits to a chunk or to append or insert additional chunks into a file that contain error correction codes for detecting and correcting errors. Such approaches may further increase the size of each chunk or the number of chunks that are ultimately synthesized into DNA [10, 17, 27].

From here on, we will not distinguish indexing bits and error correction bits from the data payload bits, and we will only consider a chunk as an arbitrary sequence of bits, $C = \{c_0, c_1, \ldots, c_{n-1}\}$, and each chunk has n ordered bits. With each chunk partitioned, they can be encoded as a strand of DNA through an encoding function $E_C : \mathfrak{C} \rightarrow \mathcal{S}$ that represents binary information on the DNA nucleotide alphabet $\Sigma = \{A, G, C, T\}$, where $\mathfrak{C}$ is the set of all possible chunks and $\mathcal{S} \subseteq \Sigma^L$ is the set of all strands of length L base pairs that represent each chunk, and we use Σ^L to denote all possible length L strands on the four-base DNA nucleotide alphabet.

We assume that the encoding process E_C adopts a code word based approach in its construction of each DNA strand $S_i \in \mathcal{S}$. That is, the encoder maintains a set of *code words* $\mathcal{C} \subseteq \Sigma^t$, where each code word is a t-nucleotide DNA-string. For each of the code words E_C maintains a bijective mapping function $f : \mathcal{C} \leftrightarrow \{0, 1\}^k$ that maps each code word uniquely to a k-bit symbol such that all k-bit strings have a corresponding code word. This mapping is also known as a codebook. Using the codebook, the encoder E_C creates each strand $S_i \in \mathcal{S}$ by dividing each corresponding chunk into a string of k-bit symbols defined as $\mathcal{Z} = \{s_0, s_1, \ldots, s_{\lceil n/k \rceil - 1}\} \mid s_0, s_1, \ldots \in \{0, 1\}^k$. Through a lookup operation in the codebook, each of the symbols $s_i \in \mathcal{Z}$ is assigned its corresponding code

[2]Storage media can alternatively be seen as an arbitrary sequence of symbols (e.g., bytes) over a fixed alphabet (e.g., bytes have 2^8 possible values), but it is more convenient for us to describe our approaches in terms of bits.

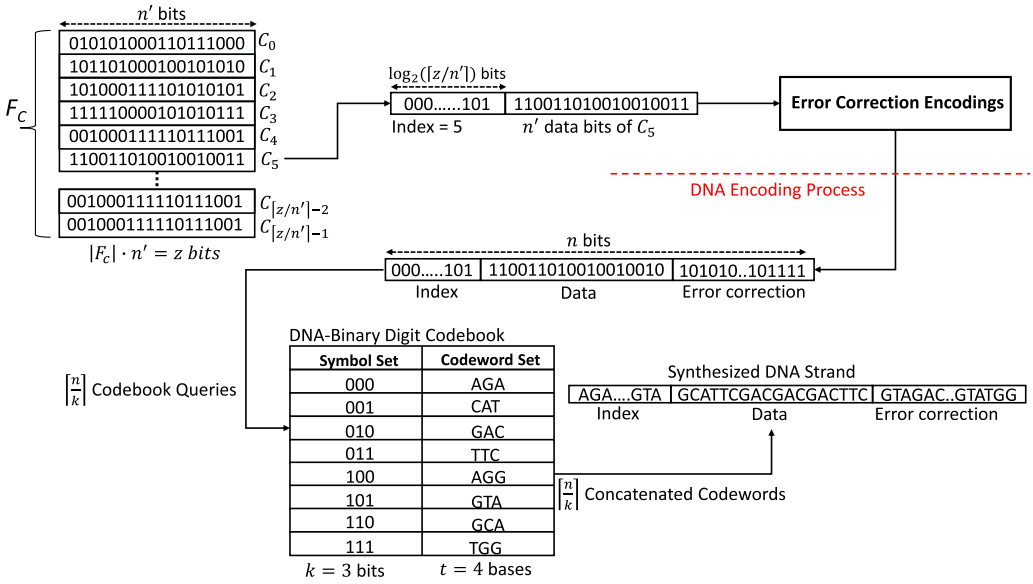


Fig. 1. Example flow of information for a code-word-based encoding process. Note, the code words of the DNA-Binary Digit Codebook are only for demonstration purposes. Not shown, but for random access, additional sub-sequences of 20 nucleotides are pre-pended and appended to the final concatenation of code words.

word from C. With each symbol assigned its code word, all code words can be concatenated to create the final strand $S_i \in \mathcal{S}$. A small example that shows the encoding process from chunks to symbols and DNA codewords is given in Figure 1. Constructions of the code word set customarily avoid problematic secondary structures such as long homopolymers that reduce sequencing accuracy and sequences used for addressing mechanisms [10, 14, 16, 17, 27, 39].

After encoding all of the strands associated with a single file, they are bookended by a pair of 20 nucleotide DNA sequences known as *primers* that are common across all strands in the same file. These primers act as addressing mechanisms that allow biochemical processes to implement random access in the sense that all strands pertaining to a certain file can be separated from the strands encoding other files [10, 23, 27, 39, 40].

2.2 Assembly Model

In a computational sense, assembly of DNA fragments for the purposes of gene assembly or assembling long DNA strings for data storage can be thought of as a concatenation operation that joins together input DNA strings in a specified order. The order is specified in the bio-chemical process by overlapping regions across sub-sequences that are Watson-Crick pairs. An example illustration of how Watson-Crick pairs drive the assembly order of DNA strands is shown in Figure 2. In this example, the DNA structure consists of three distinct regions as indicated by three unique colors on each initial fragment. The inner black double strand region represents either a sub-sequence that may have been assembled in a previous reaction or a primitive DNA code word representing some symbol of k-bits as described in the previous section. On each side of the double strand are single-stranded regions, denoted as o_i, which drive the ordering of assembly. Each single strand region o_i, known hereafter as *overhangs*, will join with its Watson-Crick complement overlapping region of the same color notated as $\overline{o_i}$. This process, known as *hybridization*, ultimately creates

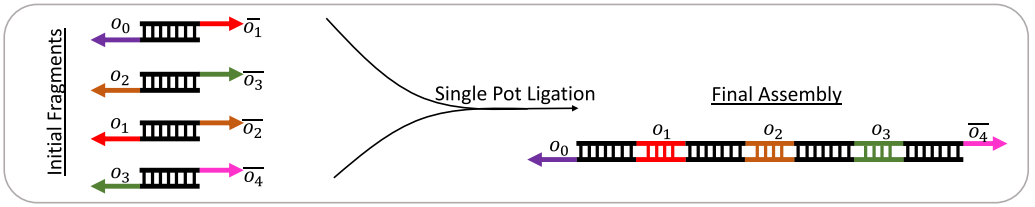


Fig. 2. Single pot assembly of DNA fragments. An initial set of fragments are mixed into a single pot reaction that assembles the strands in an order dictated by the overhangs at the end of each fragment, e.g., the first fragment in the assembly has overhangs o_0 and $\overline{o_1}$ ($\overline{o_1}$ indicates the complementary sequence of o_1).

a double strand of DNA. After all complementary overhangs join, the complete final assembly is constructed. This final assembly will have two overhangs itself such that it can be used as subassembly in a subsequent reaction. Note that, since the black region is double stranded, it is blocked from participating in anymore hybridization, and thus the single stranded overhangs are the sole determiner of ordering. We refer to a set of all possible overlapping regions that can be used in the process of assembly as O and the cardinality of the set as $|O|$.

Because overhangs dictate the ordering of fragments within a strand, repeating an overhang sequence within a reaction will result in an ambiguous ordering amongst fragments and create undesired product. It follows that there is a limit on the length of a strand that can be assembled in a single pot. To construct strands of arbitrary length, hierarchical approaches have been developed to use multiple separate intermediate reactions whose outputs are combined to create longer strands [22, 36, 37]. When designing sets of overhangs, they must be different enough to ensure that there is no non-specific hybridization. If this occurs, like the case of repeating an overhang, then there will be an ambiguous ordering of information. It has been shown in Reference [30] that overhang sets can be constructed such that non-specific hybridization is negligible. Hence, we do not consider effects of non-specific hybridization in our models, but they are an important consideration in the design of such a system.

To this point, we have described a general model that just uses hybridization to concatenate smaller strings together to generate a longer output. In a biological context, one last step is needed to permanently join the fragments together to create one single long linear DNA strand. This can be implemented by designing the overlapping sequences to be four-base sequences that facilitate ligation reactions using a ligase enzyme as is done in Golden Gate Assembly approaches [22, 30, 36, 37], or by designing longer overhang sequences so they can be used as start points for **polymerase chain reaction (PCR)** as is done in CPEC [31]. However, in all of these approaches, the hybridization of overlapping Watson-Crick sequences is all that is needed to understand the outcome of an assembly. So, we will not cover all of the requisite biochemistry, and we direct the reader to the referenced material.

3 DATA-INSPIRED OLIGO SYNTHESIS FOR DNA DATA STORAGE

3.1 Overview

Figure 3 gives an illustration of the simplest DINOS configuration using a binary code and three overhangs. As in Section 2.2, we assume that assembly is governed solely by hybridization determined by exactly matching Watson-Crick pairs. With only three overhangs, we can at most assemble two oligos at a time to ensure that there is no ambiguity of assembly from repeated overhangs. We refer to such reactions that have no ambiguity in how overhangs should assemble as *well formed* reactions. Two overhangs are reserved for connecting with adjacent reactions, and one is reserved to connect the oligos together. Hence, we form a binary assembly tree, wherein each

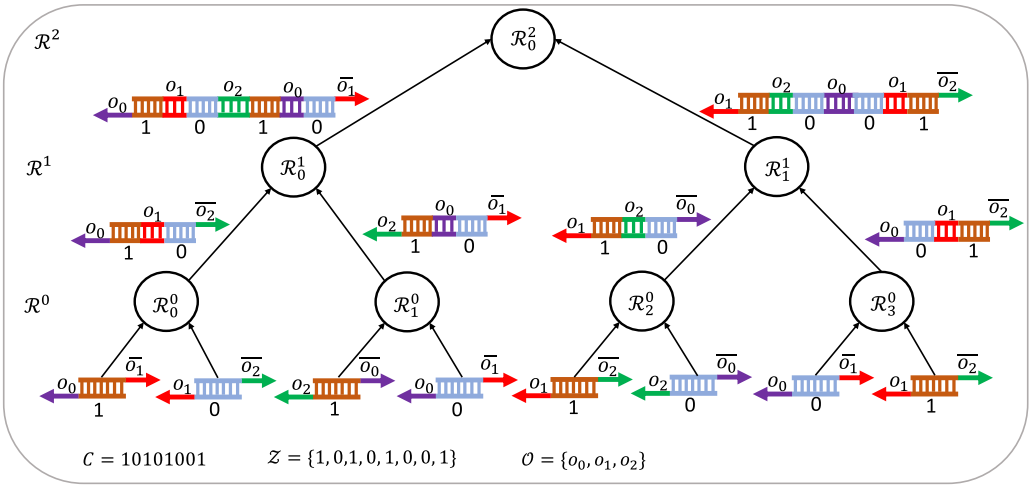


Fig. 3. An 8-bit chunk $C = 10101001$ is broken into a set of 1-bit symbols $\mathcal{Z} = \{1, 0, 1, 0, 1, 0, 0, 1\}$. Each symbol is converted into a DNA code word that is double-stranded (ds), and each code word is assigned single-stranded (ss) overhangs at each end. This ss-ds-ss DNA structure is a primitive block. Overhangs are assigned in a cyclic manner that ensures each code word can be connected with adjacent ones. Code words form groups of two from left to right to create the first layer of reactions in the tree. The grouping process continues until the entire 8-bit assembly is achieved.

reaction takes two oligos and produces a single product. We convert a chunk of 8 bits into a data strand by assembling eight 1-bit code words. Let $\overline{o}$ denote a complementary sequence to o. Starting from the first code word, we assign overhangs to each end of the code word in a cyclic manner, ensuring that the overhang between two adjacent code words are complementary. For example, the first code word will have left and right overhangs o_0 and $\overline{o_1}$, respectively, and the second code word will have left and right overhangs o_1 and $\overline{o_2}$. When the last overhang, o_2, appears as the left overhang at the third code word, the right side wraps around back to $\overline{o_0}$.

Our arrangement of overhangs has an important key property, namely, it ensures that the entire assembly tree is *well formed*. Not only are the first set of reactions trivially *well formed* due to the cyclic assignment of overhangs, but all reactions above them are also *well formed*. The cyclic arrangement of overhangs ensures that the proper permutation of overhangs is used each reaction such that an overhang is never repeated after grouping two reactions for the next layer of the tree. Furthermore, since each reaction will inherit the left and right overhangs of their left and rightmost children, respectively, the overhang assignment ensures every adjacent reaction will have proper overhangs that can join their products together.

In this system, we only need six basic DNA structures to assemble strands that represent arbitrarily long permutations of symbols. We refer to these basic structures as *primitive blocks*. There are only three ways that overhangs can be paired, and we need all combinations of code words and these three pairs, resulting in six. For this case, the number of primitive blocks needed is equal to the product of the number of code words and number of overhangs. Next, we generalize this result beyond binary symbols and a set of three overhangs.

3.2 Assembly Algorithm and Primitive Set Construction

We extend the process in Figure 3 to arbitrary overhang sets, strand lengths, and code word sets. Recall from Section 2.1 that we assume an encoding process that breaks chunks of bits

into k-bit symbols such that a chunk can be represented by a multiset of binary symbols: $\mathcal{Z} = \{s_0, s_1, \ldots, s_{\lceil n/k \rceil - 1}\} \mid s_0, s_1, \ldots \in \{0,1\}^k$. Also recall that we assume that the encoder maintains a set of DNA strings C such that each string maps uniquely to each k-bit symbol.

Now, we define the set of *primitive blocks*, $\mathcal{U}$, that we use to construct arbitrary sequences. Because we use a cyclic overhang assignment, there are $|O|$ distinct left and right overhang pairs. Therefore, we need $|O|$ versions of each code word for each possible overhang pair. Formally, this set is described as $\mathcal{U} = \{ o_j \cdot c \cdot \overline{o_{(j+1) mod |O|}} \, \forall c \in C \mid o_j \in O, 0 \leq j < |O| \}$, where $\cdot$ is used to explicitly identify the concatenation of the left overhang, code word, and right overhang. Specifying the primitive block like this abstracts away the underlying structure, e.g., a two-strand structure like Figure 2. However, it still conveys the information that an assembly approach must have two complementary versions of an overhang to allow construction to occur. Now, for each symbol in the set $\mathcal{Z}$ that represents a chunk, we assign the primitive block from $\mathcal{U}$ that has the appropriate encoding portion and overhang pairs based on the symbol's position in the set. This in turn creates an ordered multiset of primitive blocks, $\mathcal{Z}_u$, that has a one-to-one correspondence with $\mathcal{Z}$, and can be expressed as $\mathcal{Z}_u = \{u_x \mid u_x \in \mathcal{U}, 0 \leq x < \lceil n/k \rceil, u_x = o_{x \mod |O|} \cdot c_x \cdot \overline{o_{(x+1) mod |O|}} \}$ where $c_x \in C$ is the DNA code word of the corresponding symbol $s_x \in \mathcal{Z}$ at position x in the ordered set $\mathcal{Z}$.

Now, we construct an assembly tree of degree $|O| - 1$. We collect the primitive blocks at the base of the tree, from left to right, into groups of size $|O| - 1$. These groups serve as the inputs to the leaf reactions at the base of the tree. This left-to-right grouping is done at each layer of the tree until the root reaction is reached and a complete tree with degree $|O| - 1$ is created. The *growth rate*, the factor by which the assembled strands grow each layer, is $|O| - 1$. Hence, the required height is $h = \log_{|O|-1}(\lceil n/k \rceil)$, and the number of reactions at layer p, $|\mathcal{R}^p|$, can be calculated as $\frac{|\mathcal{Z}|}{(|O|-1)^{p+1}}$, assuming $p = 0$ for the leaf layer.

Because reactions in the tree only require a topological ordering, all reactions at layer p, $\mathcal{R}^p$, can be performed in parallel. Therefore, the time complexity of assembling a strand will be $O(\log_{|O|-1}(n))$ compared to $O(n)$ for sequential *de novo* approaches. Furthermore, the number of nodes in a q-ary tree is known to be $\frac{q^h - 1}{q-1}$, so it follows from our definition of height that the space complexity for synthesizing a single strand is $O(n)$ for DINOS compared to $O(1)$ for *de novo* synthesis assuming that strands can grow longer without increasing the required space.

Figure 4 illustrates the complete process of implementing four different configurations of our algorithm to assemble the same 32-bit chunk. First, the chunk is divided into symbols with the size of $k = \log_2 |C|$ bits. Each symbol is used to generate a 2-tuple where the first value represents the symbol (base-10 in Figure 4) and the second part of the tuple describes the symbol's position calculated as $i \pmod{|O|}$ for the ith symbol in the chunk. With the tuples generated for the chunk, each tuple is used to look up the primitive block in the codebook (top left table for each configuration). These primitive blocks from the codebook are then organized into leaf reactions as indicated by the table on the right of each configuration, where L_j is the jth leaf from the left of the base of the tree. With the primitive block sets constructed for each leaf, the assembly tree is constructed with a certain height, degree, and reaction count dictated by the configuration. Clearly, each different configuration leads to a different number of primitive block sets per leaf reaction, changes the height of the reaction tree, changes the number of reactions, and leads to different DNA strand (codebook) sizes. We investigate the cost and space implications of each of these configurations along with others in Section 5.

3.3 Formalizing DNA Fragment Assembly

In this section, we provide rigorous definitions for terms used to construct our assembly algorithm. Furthermore, once these definitions are described, we are able to formally define the exact problem

Chunk = 00001000010111101101001000010011

Tuple Lists : (symbol value, symbol index mod $|\mathcal{O}|$)

$|\mathcal{O}| = 3, |\mathcal{C}| = 2$: (0,0), (0,1), (0,2), (0,0), (1,1), (0,2), (0,0), (0,1), (0,2), (1,0), (0,1), (1,2), (1,0), (1,1), (1,2), (0,0) ...

$|\mathcal{O}| = 5, |\mathcal{C}| = 2$: (0,0), (0,1), (0,2), (0,3), (1,4), (0,0), (0,1), (0,2), (0,3), (1,4), (0,0), (1,1), (1,2), (1,3), (1,4), (0,0) ...

$|\mathcal{O}| = 3, |\mathcal{C}| = 4$: (0,0), (0,1), (2,2), (0,0), (1,1), (1,2), (3,0), (2,1), (3,2), (1,0), (0,1), (2,2), (0,0), (1,1), (0,2), (3,0)

$|\mathcal{O}| = 5, |\mathcal{C}| = 4$: (0,0), (0,1), (2,2), (0,3), (1,4), (1,0), (3,1), (2,2), (3,3), (1,4), (0,0), (2,1), (0,2), (1,3), (0,4), (3,0)

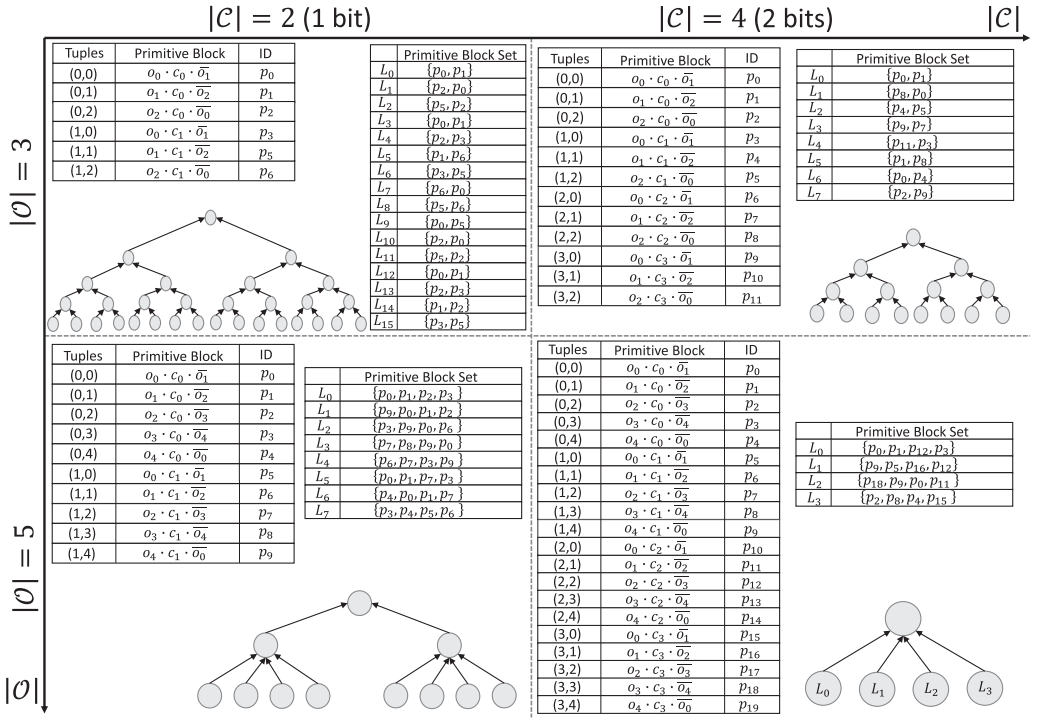


Fig. 4. Four applications of the assembly algorithm with varying code word and overhang set sizes.

that our assembly algorithm targets. We begin by describing the basic building block of assembly synthesis, the *primitive block*.

Definition 3.1 (Primitive Blocks). A primitive block p is a DNA string on the alphabet Σ that can be represented by the concatenation, denoted as $A \cdot B$ for two strings A and B, of three distinct sub-strings: $p := o_L \cdot c \cdot \overline{o_R} \mid c \in C, o_L \in O, o_R \in O, o_L \neq o_R$. Where o_L and o_R represent the left and right overhangs that facilitate its connection with other primitive blocks.

In other words, *primitive blocks* are the most basic building block of assembly, since they are individual code words that encode each $\{0,1\}^k$ of the primitive k-bit symbols. Thus, these blocks serve as the starting point of assembly and are joined together by reactions that perform concatenations. The definition for such *reactions* aims to be as abstract as possible to allow any biochemical process that fits these descriptions. We start with a definition for a special concatenation operation, referred to as *hybridization concatenation*, which reflects the product produced by the hybridization of overlapping Watson-Crick pairs as shown in Figure 2.

Definition 3.2 (Hybridization Concatenation). Let $A := o_w \cdot a \cdot o_x$ and $B := o_y \cdot b \cdot o_z$ be two DNA strings with $A[L] := o_w, B[L] := o_y$ and $A[R] := o_y, B[R] := o_z$ denoting each of the strings'

left and right substrings, respectively, in the concatenation. Let $\varnothing$ denote the empty string. The hybridization concatenation of the two strings, denoted with the symbol $\otimes$, is defined as $A \otimes B := o_w \cdot a \cdot o_x \cdot b \cdot o_z$ iff $A[R] = \overline{B[L]}$ and $A \otimes B := \varnothing$ otherwise. The hybridization concatenation operation output can be defined for more than two strings, say, $A \otimes C \otimes B$, as long as $A[R] = \overline{C[L]}, C[R] = \overline{B[L]}$.

There are several characteristics to point out about hybridization concatenations. One, the output of the operation is only defined if the right and left overhangs for the first and second strings, respectively, are Watson-Crick complements.[3] This captures the fact that two overlapping regions will not join together due to a lack of Watson-Crick pairing. Second, since the overhang regions necessarily start as single-stranded DNA to enable connections, the hybridization turns these regions into double-strand segments after hybridization (Figure 2).

Definition 3.3 (Reaction). Given an ordered set of DNA strings $D = \{d_0, d_1, \ldots d_{N-1}\}$ where each string d_i has a left and right overhang denoted by indexing as $d_i[L] \in \mathcal{O}$ and $d_i[R] \in \mathcal{O}$, a reaction R takes as input D and produces an output DNA string that is a hybridization concatenation of all DNA strings in D: $R(D) := d_i \otimes d_j \otimes d_k \ldots \otimes d_l \mid i := \sigma(0), j := \sigma(1), k := \sigma(2), \ldots l := \sigma(N-1)$. Where σ is a permutation function that maps the indices of the hybridization function to the indices of the input set D of strings such that the definition of hybridization concatenation is satisfied. For example, $\sigma(0)$ maps the first string in the operation to some index $[0, N-1]$ that represents an input string d_i.

A major component of a *reaction* is being able to define what the permutation function σ is. This function defines the output string as it determines the positioning of each input string d_i in the series of hybridization concatenations that define the output $R(D)$, and is driven solely by the overlapping regions at each end of each input string. Since we are interested in deterministically producing one unique product for a given reaction, there must exist one unique function σ for a given set of input strings D. If there is, however, more than one possible σ that may be defined for a set D, then there is more than one possible arrangement that will occur with high probability when biochemically implemented. This will also occur if there does not exist a maximal σ that includes all of the $d_i \in D$ input strings in the set of hybridization concatenations. The existence of a maximal and unique σ leads us to definition of a *well formed reaction.*

Definition 3.4 (Well Formed Reactions). A reaction R is said to be well formed if there exists, for some input of strings $D = \{d_0, d_1, \ldots d_{N-1}\}$, a unique permutation function σ for R such that all N input strings can be hybridization concatenated with an output that is not the empty string.

The existence of a permutation function σ depends on the overhangs of each input string, as these determine the possible biochemical interactions that may occur. The following theorem states the overhang requirements for a set D to ensure a maximal unique σ, along with a maximum value for the size of the input set D given a starting overhang set $\mathcal{O}$. From this point on, it is assumed that each input string in D must have a left and right overhang.

Theorem 3.1. *A reaction R is well formed for an input set of N strings, D, if and only if across all input strings $d_i \in D$ there are exactly $N+1$ unique overhang strings that form a set $\mathcal{Y}$ such that there are $N-1$ unique overhang strings that form a set $\mathcal{X} \subset \mathcal{Y}$ where for each overhang $o_\mathcal{X} \in \mathcal{X}$, $o_\mathcal{X}$ appears exactly once as a right overhang for d_i and a left overhang for d_j when $i \neq j$. In other words, for an overhang set $\mathcal{O}$ that has $|\mathcal{O}|$ unique overhang strings, a reaction R may have at most only $|\mathcal{O}| - 1$ possible input strings.*

[3] In a real reaction, it would be important to design overhang sequences to have a much higher affinity for on-target binding as compared to off-target binding.

Proof. We first show that a *well formed* reaction implies the existence of sets $\mathcal{Y}$ and $\mathcal{X}$, and then show that if sets $\mathcal{Y}$ and $\mathcal{X}$ exist then R must be *well formed*. By the definition of *well formed*, for the input set of strings D there must be a unique permutation such that the hybridization concatenation of all N strings is defined so the output is not the empty string. For this to occur, by definition of the *hybridization concatenation* operation, each string and its neighbors in the sequence of hybridization concatenations for reaction R must having matching overhang strings. For example, let i denote the index representing a string's positioning in the sequence of hybridization concatenations for R, then $d_{i-1} \otimes d_i \otimes d_{i+1}$ and $d_{i-1}[R] = \overline{d_i[L]}, d_i[R] = \overline{d_{i+1}[L]}$. Therefore, there are $N-1$ unique overhangs that are shared between each string, and this set of overhangs forms $\mathcal{X}$. Since the reaction is assumed to be *well formed*, the permutation must be unique, and thus each overhang must occur exactly once as a left and right overhang. Last, since the first and last strings in the sequence of hybridization concatenations do not have left and right neighbors, respectively, these overhangs are not shared with any other string in D and thus are two more unique overhangs that can be added to $\mathcal{X}$ to construct $\mathcal{Y}$.

However, if across all input strings $d_i \in D$ there is a set of $N+1$ unique overhangs to construct set $\mathcal{Y}$, then we can allow two overhangs to be allocated to the strings in D that will be placed as the first and last arguments for R's hybridization concatenation. Thus, given that the remaining $N-1$ overhangs that constitute $\mathcal{X}$ all occur exactly once as a left and right overhang, there is only exactly one permutation of input strings such that the series of hybridization concatenation operations are not $\emptyset$. It is also easy to see that for $N-1$ shared overhangs, the hybridization concatenation of all N can be achieved. Thus, the reaction is well formed for this input. This concludes the proof, and it follows that if a set of unique overhangs O of size $|O|$ exists, then there can only ever be at most $N+1 = |O| \implies N = |O| - 1$ input strings. □

By defining *well formed reaction* and how to construct a set of input strings to a reaction so it is *well formed*, and by observing that the output strings of reactions can be used as input strings to subsequent reactions, since each output string has a left and right overhang as determined by the corresponding overhangs of the first and last strings in the reaction's hybridization concatenation sequence, the definition of a *well formed reaction network* can be constructed.

Definition 3.5 (Well Formed Reaction Networks). A *well formed reaction network* is a directed acyclic graph of reactions $G(V, E)$ with a set of vertices V representing the reactions of the network, and edge set E where each edge represented by an ordered pair $(v_i, v_j) \mid v_i \neq v_j$ indicates that reaction v_i produces a string used as input in reaction v_j. Such a network must have a set of leaf reactions in which their input strings are primitive blocks, a set of intermediate reactions with input strings of other intermediates or leaf reactions, and a single terminating reaction that constructs the final target assembly. A terminating reaction may be a leaf reaction where its inputs are primitive blocks, or such that its inputs are produced by intermediate reactions. A reaction network is *well formed* if for each reaction is *well formed* itself.

Finally, this leads us to the definition of the precise problem statement that our algorithm addresses:

Definition 3.6 (DNA Strand Assembly Problem). Define a well formed reaction network R_N and a set of primitive blocks $\mathcal{U}$ such that any possible permutation of code words in C of any length can be assembled in the order as specified by the symbols of any arbitrary chunk of data C.

Next, we show that our assembly algorithm satisfies the DNA Strand Assembly Problem.

3.4 Proof of Correct Assembly

Theorem 3.2. *Any arbitrary tree of height h and degree $|O| - 1$ can be implemented with $|O| \geq 3$ overhangs, given a sufficient overhang assignment that does not repeat any overhang across all children of some parent node y.*

Proof. We use proof by induction. Let h denote the height of a tree, relative to the root node. We start with a base case: a tree with $h = 1$. The root node has $|O| - 1$ primitive block children. Each child needs two bookend overhangs, but adjacent children share an complementary overhangs to ensure assembly. Furthermore, all overhangs must be unique to avoid improper assemblies. This is equivalent to each child having a unique overhang on the left, and the rightmost child also having a unique overhang on the right. So, $|O| - 1$ children use $|O| - 1 + 1 = |O|$ overhangs. Thus, a tree of $h = 1$ can assemble $|O| - 1$ children. Now, we will use this base case to show that if a tree of height h and degree $|O| - 1$ can use $|O|$ overhangs, then a tree of height $h + 1$ is also an assembly tree that uses $|O|$ overhangs.

For the inductive step, consider the tree of height h. For each leaf at this h layer, we can add $|O| - 1$ children to complete the $h + 1$ level. Let us take one node from the h level that we extend, call it y. It is now the parent of $|O| - 1$ nodes. Its left-most child would share y's left overhang. The right-most child must share y's right overhang. This ensures proper assembly at level h.

We know that the sub-tree centered at y is equivalent to the base case and can therefore be assembled with $|O|$ overhangs. Once assembled, only y's left and right overhangs will be exposed. Hence, we need only assign overhangs for each of y's children in such a way as to ensure each overhang is used once and that the y's left and right overhangs only appear at the left-most and right-most overhangs of the first and last child. We can easily do this, since we have $|O|$ unique overhangs. We can repeat this process for all parent nodes at level h to make $h + 1$. Finally, since the simplest assembly of two strands requires two unique end overhangs and a unique overhang to join them together, $|O|$ must be at least 3. □

In other words, starting from the root, each node can be treated like a leaf node. Given a node with an arbitrary left and right overhangs, $|O|$ overhangs can be used to ensure the assembly of its $|O| - 1$ children. This process can be repeated down the tree until the overhangs of each code word are specified, thus specifying the set of primitive blocks needed to assemble a complete strand. We are now interested in knowing an overhang assignment that minimizes $|\mathcal{U}|$, the size of the primitive block set.

The size of the primitive block set for tree assembly is upper bounded by $(|O|^2 - |O|) \cdot 2^k$ and lower bounded by $|O| \cdot 2^k$. The upper bound arises from a naive overhang assignment that allows for any two unique overhangs to be assigned to the left and right overhangs of a child. Thus, given the fact an overhang cannot repeat such that the left and right overhangs are equal, there are $(|O|^2 - |O|)$ possible overhang pairs for each code word. The primitive block set cannot be lower than $|O| \cdot 2^k$, because there needs to be at least $|O| - 1$ versions of a given code word to assemble a $|O| - 1$ long repetition of the same code word in a leaf reaction. Furthermore, the fact that the right overhang is unique in the leaf reaction implies that there is at least one more version of the code word with this overhang as its left overhang. This other version would be required to to facilitate the assembly of a $|O|$-run of a code word, because the assembly of the first $|O| - 1$ would need to be subsequently assembled with $|O|$-th code word.

Theorem 3.3. *$|O|$ ordered overhangs arranged sequentially in a repeating pattern across the bottom of an $|O| - 1$ degree tree will always generate primitive blocks that result in a well formed reaction network for any sequence of symbols $\mathcal{Z}$ and minimizes $|\mathcal{U}|$.*

Proof. It is well known that for any $n \in \mathbb{Z} \mid n > 1$, n and $n + 1$ are coprime. Hence, $|O|$ and $|O| - 1$ are coprime. Furthermore, without loss of generality, we can lay out the overhangs as an ordered set of magnitude $|O|$ and simply refer to them by their index, 0 to $|O| - 1$. These indices form the finite cyclic group $\mathbb{Z}_{|O|}$, e.g., the set of remainders under *mod*-$|O|$ addition.

It is also well known that if a number n is coprime to m, then addition by n is a generator of the finite cyclic group $\mathbb{Z}_m$. By repeatedly adding n modulo m, all members of $\mathbb{Z}_m$ will be enumerated before a repeat is observed.

The leaf level of the tree has a stride of 1 that is trivially coprime with all integers. Therefore, addition by 1 will generate all $|O|$ overhangs of O without repeat. This guarantees proper assembly of each reaction at the leaves.

The next level of the tree is equivalent to a stride of $(|O| - 1)^l \mid l > 1$, where l is the level, counted from the bottom of the tree. Since $(|O| - 1)^l$ must also be coprime with respect to $|O|$, addition by $(|O| - 1)^l$ modulo $|O|$ will also generate all $|O|$ overhangs without repeat. This proves that a sequential group of $|O| - 1$ reactions at level l will assemble properly, since it satisfies that each overhang from O occurs exactly once, and every consecutive reaction will have appropriate complementary overhangs due to the complementary assignment at the bottom of the tree.

Because each level of the tree is a power of $|O| - 1$ and coprime to $|O|$, all levels of the tree meet this criteria and will assemble properly. Last, since the assignment of overhangs to code words is cyclic, there are only $|O|$ possible overhang pairs and thus only $|O|$ versions of each code word required. This achieves the previously mentioned lower bound. □

4 ELIMINATING REDUNDANT REACTIONS

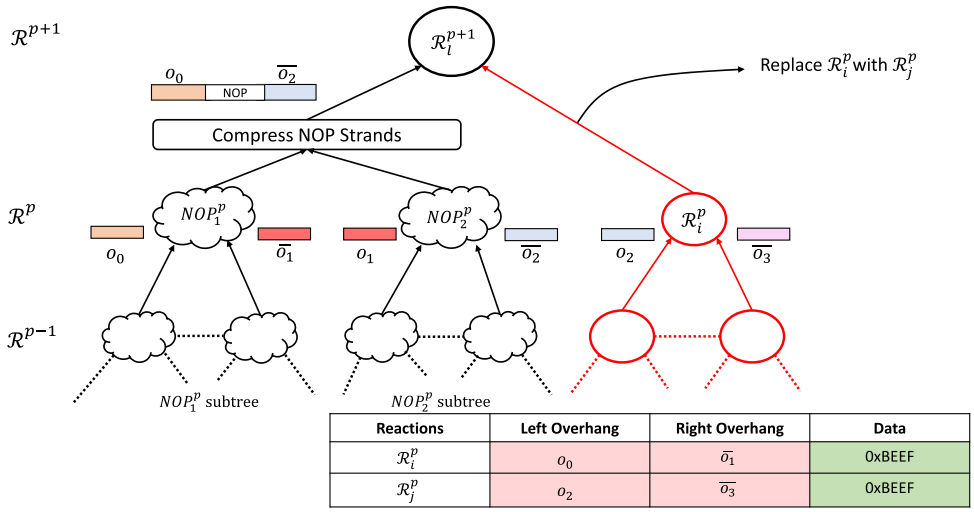
Due to the inherent redundancy found in information, it is likely that there will be many redundant reactions that assemble the same data. This redundancy offers opportunities for decreasing the costs of synthesis by sharing the output of a reaction amongst more than one parent node. We refer to this as *coalescing*, since we are literally joining the two reactions to happen at the same time, thereby eliminating one of them from the assembly tree.

Coalescing two reactions requires two criteria to be met. First, the two reactions must assemble the same data sequence. Note, we can search for matching data sequences at any level of the assembly tree. Intuitively, we will expect to find more repetition near the leaves of the tree where there are fewer possible data sequences than near the root of the tree. Second, the strands produced from both reactions must have the same bookend overhangs. The second requirement is important to respect, as replacing a reaction with another that has different overhangs will result in a reaction that is not *well formed*. This is because in each reaction all $|O|$ overhangs occur exactly once and replacement of any reaction by another with different overhangs will result in duplicated overhangs at the next level in the tree, leading to multiple fragments competing at a common overhang site.

In a data storage system that is processing large files or many files at the same time, we need not restrict the search for redundancy to a single strand. Redundancy can be eliminated across strands within a record or across records. This is because reactions can be viewed as producing generic sequences of information, and each strand produced by a reaction does not inherently include any specific strand or record identifier like primers that are used for random access [10, 27, 39, 40]. In fact, sequences like primers do not need to be added to a strand until the root node is assembled, allowing all products produced under the root to be shareable to any other strand in any other record.

4.1 Optimization Techniques

We identify three different optimization techniques for coalescing reactions. The *ideal* optimization allows for two reactions to be coalesced if they produce the same data even if they do not necessarily share the same bookend overhangs. This serves as a lower bound on the number of



Reactions	Left Overhang	Right Overhang	Data
$\mathcal{R}_i^p$	o_0	$\overline{o_1}$	0xBEEF
$\mathcal{R}_j^p$	o_2	$\overline{o_3}$	0xBEEF

Fig. 5. Reactions $\mathcal{R}_i^p$ and $\mathcal{R}_j^p$ produce the same data, 0xBEEF, but have different left and right overhangs when $|O| = 4$. The left and right overhangs of $\mathcal{R}_i^p$, $(o_0, \overline{o_1})$, are converted to the left and right overhangs of $\mathcal{R}_j^p$, $(o_2, \overline{o_3})$, by inserting two *NOP* reactions at level p in the tree. The *NOPs* move the overhangs of $\mathcal{R}_i^p$ along the cyclic overhang assignment until the desired overhangs are reached. Once reached, the output of $\mathcal{R}_j^p$ can replace $\mathcal{R}_i^p$. The adjacent *NOPs* with left and right overhangs $(o_0, \overline{o_1})$ and $(o_1, \overline{o_2})$ can be compressed into a single *NOP* with overhangs $(o_0, \overline{o_2})$ if such a strand is included in the primitive block set.

reactions that are needed to synthesize some data. This technique is not implementable in a real system but is useful for analysis. We also consider an optimization that searches through all of the reactions of an un-optimized assembly tree and coalesces all reactions that share the same overhangs and produce the same data. We call this base optimization *base-opt*. In our studies, a significant gap exists between the *ideal* and *base-opt*, and to address this, we study an alignment technique to facilitate more reaction coalescing.

4.2 Alignment Padding

To facilitate more occurrences of reactions producing the same data with matching overhangs, we develop an alignment technique that trades off storage density and primitive block set size for reaction elimination opportunities. The key idea of alignment padding is that we can leverage the cyclic assignment to adjust overhangs of an arbitrary reaction by inserting padding to its left, thereby shifting overhang assignment and creating a match that enables coalescing. We refer to such padding as a ***no-operation (NOP)*** reaction, since it represents no symbol. Figure 5 illustrates the idea by inserting *NOP* reactions into the assembly tree to shift a reaction's overhangs to align with the overhangs of another reaction producing the same data. These *NOP* reactions can be thought of as a subtree that has reactions that only assemble *NOP* strands. All reactions to the right of the NOP will be shifted by the same amount as the target reaction.

To the level of the tree of which padding reactions are added, they are interpreted like any other reaction. So, if *NOP* reactions are added to layer p, then there must be an appropriate number of reactions at layer $p + 1$ to handle the additional load. Thus, *NOP* padding adds reactions to the tree. However, the *NOP* reactions of multiple alignment transformations can be effectively bundled together, limiting the number of new reactions added at layer $p + 1$. Because reactions

group $|O| - 1$ children, if multiple alignment transformations collectively add less *NOP* reactions than $|O| - 1$, then only one additional reaction is needed at $p + 1$. Thus, for the overhead of one additional reaction, multiple reactions can be transformed to be redundant and optimized out. Note, this bundling is only possible if the padding reactions for multiple transformations are inserted for data that are assembled on the same strand. This is because insertions on the assembly tree of one strand are not visible to the grouping process of an assembly tree for another strand of data. So, insertions must be applied appropriately so the overall number of reactions do not increase.

To avoid such situations, we propose a heuristic based on two metrics: distance and re-use count before padding. Distance is the number of *NOPs* that must be inserted to transform the overhangs of certain reactions. Placing an upper bound on the distance helps avoid using too many *NOPs* for any one redundancy operation and allows more redundancy operations to be bundled together in a reaction. Through experimentation, we found that the distance must be less than $(|O|-1)/4$. This means that each padding operation at most takes up 1/4 of the capacity of a parent node. Given enough opportunity in a single strand, at least four padding transformations can be amortized for a single overhead parent reaction.

We also consider the number of times a reaction is re-used in the *base-opt* case. The reason to consider this count is that it reflects the amount of times that a single transformation will have to be applied. Applying a transformation to an already highly redundant *base-opt* reaction may simply add overhead. Through experimentation, we determined that if the re-use count exceeds a threshold of 10, then it is already sufficiently redundant, and we do not perform any alignment padding, since it may only increase reactions.

There are also several optimizations that can be applied to *NOP* trees so reaction count increases and density loss is limited. For *NOP* reactions, no physical reaction must be implemented. Instead, only a single *NOP* primitive block need be inserted at the root of the *NOP* sub-tree. For example, from Figure 5, the *NOP* sub-trees for NOP_1^p and NOP_2^p can be replaced by *NOP* primitives with (left,right) overhang pairs $(o_0, \overline{o_1})$ and $(o_1, \overline{o_2})$, respectively. From Corollary A.1, we show that there are two unique possible rotations through the set of overhangs at any level in the tree. So, to have enough *NOP* strands for each rotation, the primitive set will need to be expanded by $2|O|$ strands. Furthermore, as shown in Figure 5, consecutive *NOP* strands in a reaction may be compressed to one single strand. However, this requires an additional $|O|^2 - |O|$ primitive blocks to accommodate all possible overhang permutations that may arise from this optimization.

5 EVALUATION

We evaluate DINOS on the Silesia Corpus that is commonly used to benchmark compression algorithms. The corpus contains a wide range of file types and sizes, as summarized in Table 1 [3]. Of particular interest in our evaluation is the influence of the configurable parameters that tune the primitive block set size on the number of reactions, opportunities for coalescing redundant reactions, density, and cost. We are also interested in comparing their benefits when applied to compressed and uncompressed data files. When comparing number of reactions and effectiveness of coalescing, we may ignore the specific chemistry and specific strand encodings, since they are orthogonal concerns.

Through all of the following experiments, the strand size is held to a constant 4,096 bits of information. We choose this strand size to have a wide range of tree heights for various overhang set sizes. The 4,096 bit strands are then divided into k-bit chunks, mapped to primitive blocks, and assembled according to the algorithm. Note, not all studied symbol set sizes and overhang set sizes will result in a complete tree of degree $|O| - 1$. We have developed efficient solutions to deal with incomplete trees and omit their discussion for space, and we point out that the solutions we used

Table 1. Compressed and Decompressed File Sizes of the Silesia Corpus with Brief File Descriptions

Silesia Corpus Files

Filename	Description	Type	Size MB	Size MB (zpaq)	Reactions	Reactions (zpaq)
dickens	Collected works of Charles Dickens	English text	10	2.1	$8.2 \cdot 10^7$	$1.69 \cdot 10^7$
mozilla	Tarred executables of Mozilla 1.0	exe	51	12	$4.12 \cdot 10^8$	$9.69 \cdot 10^7$
mr	Medical magnetic resonance image	picture	10	2.2	$8.02 \cdot 10^7$	$1.76 \cdot 10^7$
nci	chemical database of structures	database	34	1.3	$2.7 \cdot 10^8$	$1.01 \cdot 10^7$
ooffice	A dll from Open Office.org 1.01	exe	6	1.8	$4.95 \cdot 10^7$	$1.42 \cdot 10^7$
osdb	OSDB MySQL database	database	10	2.2	$8.11 \cdot 10^7$	$1.77 \cdot 10^7$
reymont	Polish pdf	Polish text	6	0.96	$5.33 \cdot 10^7$	$7.7 \cdot 10^6$
samba	Tarred source code of Samba 2-2.3	src	21	3.1	$1.74 \cdot 10^8$	$2.46 \cdot 10^7$
sao	The SAO star catalog	bin data	7	3.9	$5.83 \cdot 10^7$	$3.14 \cdot 10^7$
webster	1913 Webster Unabridged Dictionary	html	41	5.7	$3.34 \cdot 10^8$	$4.56 \cdot 10^7$
xml	XML files	html	5	0.33	$4.3 \cdot 10^7$	$2.63 \cdot 10^6$
X-ray	X-ray medical picture	picture	8	3.7	$6.82 \cdot 10^7$	$2.95 \cdot 10^7$

do not affect the well-formed property of our reactions nor how redundant reactions are found and coalesced.

We also evaluate the effectiveness of our redundancy optimizations. Since compression algorithms will undoubtedly remove redundancy, decrease the size of the file, and ultimately decrease the number of reactions that are needed, we are also interested in studying opportunities for removing redundant reactions using our techniques both before and after applying compression to the data. We choose a high-compression ratio algorithm for file archiving, *zpaq* [25], which has higher compression ratios than typically available compression tools such as *zip*, *bzip*, *7zip* [26]. Using *zpaq*, with its highest compression option (*m5*), we also compress each of the Silesia Corpus files and study the optimization opportunities therein. We report the size after compression in Table 1. Although redundancy can be looked for across files, we limit the scope of redundancy to each individual file.

In Sections 5.5 and 5.6, we compare the cost and density of DINOS to state-of-the-art synthesis and DNA assembly approaches. For those analyses, we select specific chemistries and strand encodings. Tuning the chemistry and DNA sequences is critical in an actual storage system, but we limit our discussion of these issues in the interest of space.

5.1 Impacts of Symbol Size and Number of Overhangs

The two parameters that drive the primitive block set size and the assembly algorithm are symbol size (k) and overhang set size. First, we consider (1 bit, 3 overhangs). This configuration requires the largest number of reactions, because it has the fewest symbols and the fewest parts are assembled in each reaction. The last two columns of Table 1 report the total reactions needed to assemble

Table 2. Geomean of Normalized Reaction Counts across All Decompressed Files in the Silesia Corpus for Overhang Set Size ($|O|$) and Symbol Length (k) Pairs

Geomean of Normalized Reactions - No Optimization

k (bits)	$\|O\|$				
–	3	5	9	17	65
1	1	0.33	0.14	$6.67 \cdot 10^{-2}$	$1.59 \cdot 10^{-2}$
2	0.5	0.17	$7.16 \cdot 10^{-2}$	$3.35 \cdot 10^{-2}$	$8.06 \cdot 10^{-3}$
3	0.33	0.11	$4.81 \cdot 10^{-2}$	$2.27 \cdot 10^{-2}$	$5.62 \cdot 10^{-3}$
4	0.25	$8.33 \cdot 10^{-2}$	$3.59 \cdot 10^{-2}$	$1.68 \cdot 10^{-2}$	$4.15 \cdot 10^{-3}$
5	0.2	$6.72 \cdot 10^{-2}$	$2.91 \cdot 10^{-2}$	$1.39 \cdot 10^{-2}$	$3.42 \cdot 10^{-3}$
8	0.12	$4.18 \cdot 10^{-2}$	$1.78 \cdot 10^{-2}$	$8.55 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$

All values are normalized with respect to the number of reactions for the pairing of 1-bit code words and three overhangs.

each file under the case of (1 bit, 3 overhangs). The case of uncompressed and compressed files are reported in columns 6 and 7, respectively. Note, no optimizations are included in these results. Since this configuration requires the largest number of reactions, we normalize all other configurations to it.

Table 2 shows the geometric mean of normalized reaction counts across all files with no optimizations applied, where all reaction counts for each *(symbol size, overhang count)* pairing are normalized to that of (1 bit, 3 overhangs). This table shows that increasing both parameters has a strong impact on the number of reactions. For example, increasing to (2,3) reduces the reaction count to half, and (1,5) reduces the reaction count by a factor of approximately 3. However, as symbol length and overhang count are increased, the effect diminishes, for example moving from (1,5) to (1,9) results in only a 2.36× reduction. This suggests diminishing returns, where increasing these parameters leads to less and less benefit and also that overhangs have a bigger impact than scaling symbol length for the same factor increase.

Last, it should be noted that scaling k exponentially increases the primitive block size, $|\mathcal{U}|$, while scaling $|O|$ only linearly scales it. Hence, $|\mathcal{U}_{(1,3)}| = 6$ while $|\mathcal{U}_{(8,3)}| = 768$. For a similar improvement in reactions as (8,3) has over (1,3), we can select (1,9) where $|\mathcal{U}_{(1,9)}| = 18$, in other words, 42× fewer primitive blocks.

5.2 Opportunities for Redundancy Elimination

Next, we evaluate the amount of redundancy that exists while assembling strands for a file. The overhang counts are varied from 3–65, and the symbol size is kept constant at 1 bit. This ensures that we study a wide range of data granularities and that will be important when understanding the effects of optimizations. Note, hereinafter, we report relative values for the reaction counts associated with each optimization. Using the last two columns of Table 1 and the data of Table 2, raw reaction counts can be computed for each configuration. Though, for convenience, we also include raw reactions for every configuration and benchmark in our supplementary document.

Figure 6(a) shows *ideal*'s reaction count normalized to no optimization and averaged using geometric mean across the corpus. Recall that *ideal* ignores overhangs and assumes that as long as the same sequence of symbols is produced that the reactions can be coalesced. The general trend is that as the number of overhangs increases, the effectiveness of optimization decreases. This is due to the increase in the granularity of information created at each layer of the tree that makes it

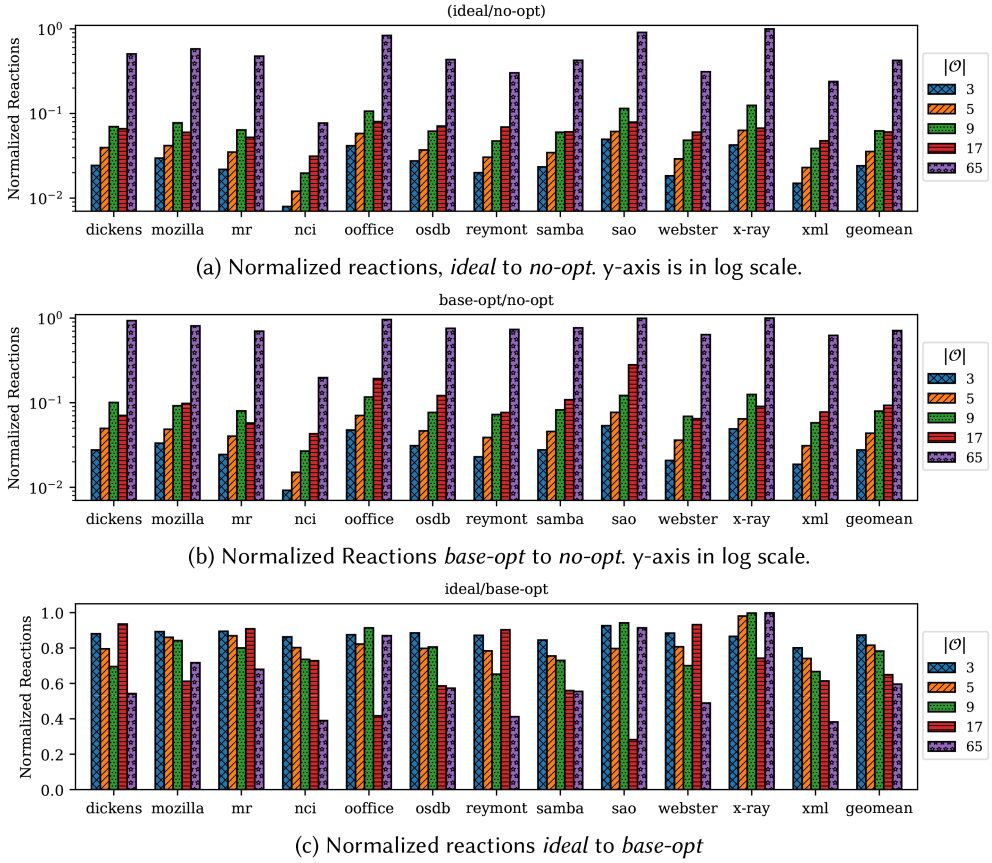
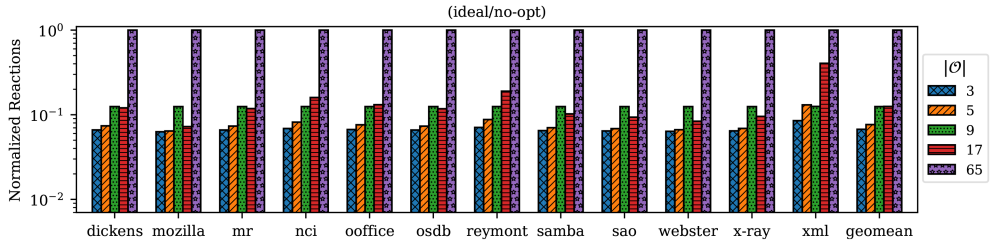


Fig. 6. Normalized reactions for each file for both *base-opt* and *ideal* optimizations. No compression is applied before constructing the assembly tree.

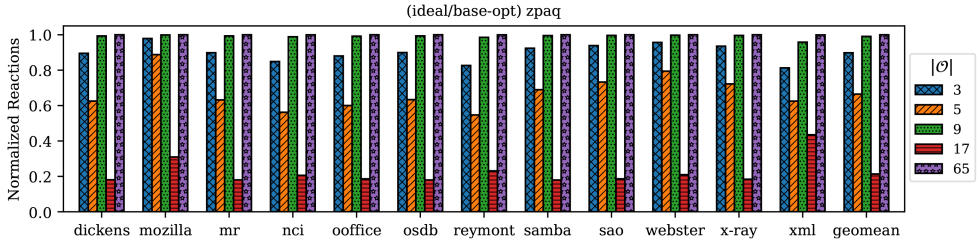
less likely to find redundant data. As expected, some files have more redundancy than others. For example, with 3 overhangs, the number of reactions for *nci* can be reduced by 99% relative to no optimization, and for 65 overhangs reactions can be reduced by 92.23%. This is in contrast to the average reduction in reactions of 97.59% and 57.6% for 3 and 65 overhangs, respectively.

Now, we compare *ideal* with *base-opt* in Figures 6(b) and 6(c). Recall that *base-opt* requires that the bookend overhangs of two reactions match in addition to their data sequences matching. Even when we take into account the restriction of matching overhangs for redundant data, there is a fair amount of redundancy available. On average, *base-opt* is able to reduce reactions by 97.23% and 28.83% for 3 and 65 overhangs, respectively. Although *base-opt* is effective at $|O| = 65$, the ideal case can remove an additional 28.77% portion of *no-opt*'s reactions. Figure 6(c) provides a direct comparison between the two optimizations by normalizing the reaction count of the ideal case to that of *base-opt*. On average, the disparity between the two grows as the number of overhangs increases. This is due to the increasing difficulty for *base-opt* to take advantage of redundancy due to a greater chance of mismatched overhangs.

The added requirement of matching overhangs effectively creates more information states that need to be assembled compared to that of *ideal*, and theoretically it can increase the number of states by a factor that is at most $|O|$.



(a) Normalized reactions, *ideal* to *no-opt*. y-axis is in log scale.



(b) Normalized reactions *ideal* to *base-opt*

Fig. 7. Comparison of reactions between *base-opt* and *ideal* optimizations when *zpaq* compression is applied before constructing the assembly tree.

For example, when $|O| = 17$ for *dickens*, there are only 2,344 unique 16-bit information states each repeated 2,174 times on average, while *sao* uses all 2^{16} unique 16-bit information states each repeated only 55 times on average. When considering overhangs, both *dickens* and *sao* see similar increases in unique states of 11.2× and 12.88×, respectively. However, since *sao* uses all 2^{16} states, the overhangs inject more total states that must be synthesized than for *dickens*. This leads to a larger relative reaction gap when comparing *ideal* and *base-opt* across these two benchmarks, e.g., *ideal* removes 71% and 6.45% of *base-opt*'s reactions for *sao* and *dickens*, respectively.

Figure 7(a) compares the number of reactions of *ideal* against the number of reactions for no optimization when compressing files with *zpaq*. Since compression removes redundancy, the ideal optimization is no longer able to find significant redundancy when $|O| = 65$, resulting in only 0.02% reduction in reactions relative to no optimization on average. However, for smaller $|O|$ that have more layers with smaller data granularities, the relative reductions for the compressed files compared to the de-compressed files are similar on average. For 2 and 17 overhangs, reductions for the compressed files are still 93.27% and 87.46% as compared to 97.58% and 93.96% for de-compressed files.

When comparing *base-opt* and *ideal* for compressed files, the largest discrepancy between optimizations occurs at 17 overhangs. On average, as shown in Figure 7(b), *ideal* can additionally remove 78.7% of *base-opt's* reactions for this $|O|$. This large difference can be explained by studying the distributions of information states for the compressed files when $|O| = 17$. We found that the compression algorithm expands the number of information states required for synthesizing a file and the variance of the frequencies for each information state is much smaller than that of decompressed files. For example, the uncompressed *dickens* had 2,344 states at the first level of the assembly tree when $|O| = 17$ with an average frequency for each information state of 2,174 with variance across the frequencies of $75 \cdot 10^6$. With compression, it used all 2^{16} possible states with an average re-use count of 16 and a variance of 17. This explains the considerable gap when compared to no compression. Also, since $|O| = 17$ leads to larger data granularities, thus more

Table 3. Geomean of Normalized Reaction Counts for Both *base-opt* and *ideal* across All Decompressed Files in the Silesia Corpus for $|O|$ and (k) Pairs

Geomean of Normalized Reactions (Decompressed) -*Ideal* & *Base-opt*

k (bits)	$\|O\|$									
	ideal					*base-opt*				
–	3	5	9	17	65	3	5	9	17	65
1	1	0.49	0.37	0.17	0.28	1	0.53	0.41	0.22	0.41
2	1	0.51	0.29	0.24	0.23	1	0.54	0.32	0.37	0.24
3	1.37	0.66	0.42	0.43	0.19	1.37	0.68	0.56	0.55	0.18
4	1	0.49	0.33	0.32	0.14	1	0.53	0.41	0.39	0.13
5	1.19	0.61	0.45	0.37	0.12	1.25	0.67	0.55	0.39	0.11
8	1	0.51	0.37	0.25	$7.66 \cdot 10^{-2}$	1	0.54	0.41	0.25	$6.84 \cdot 10^{-2}$

All values are normalized with respect to the number of reactions for the pairing of 1-bit code words and three overhangs for each optimization.

information states that get multiplied by a large multiplier, this leads to larger relative increases in reactions as compared to smaller $|O|$ during compression. This effect is similar to why *dickens* behaved better previously with *base-opt* for decompressed data compared to *sao*.

Interestingly, when comparing the total number of reactions for *base-opt* both before and after compression, there are overhang set size configurations that result in smaller reaction counts for decompressed files. This tends to occur primarily for $|O| = 17$ and for decompressed files that tend to have a small number of information states at this granularity, e.g., *mr*. We find in this case that decompressed version of *mr* has 60% less reactions than its compressed counterpart, the largest of any record studied in this work. This implies that the large scale of reaction coalescing afforded by the low entropy of information states for certain files and granularities can be more effective for reducing reactions rather than increasing the entropy and reducing the absolute amount of data to be synthesized with decompression.

We also compared the number of reactions before and after optimization for two files filled with random data. One with approximately as many bytes as the average file size before compression (17 MB) and another smaller random file of size equal to the average file size of compressed data (3 MB). We found that *ideal* optimization reduces the reactions by nearly the same amount compared to compressed files with similar size. The reason reactions can be easily removed from random data in some cases is because for small granularities (<16 bits), it is inevitable that they will be repeated for the file sizes we study. This also indicates the compression used here is approximate to random files in terms of entropy, which makes sense given the larger number of states with lower variance mentioned before for *dickens*. The data for random files can be found in the supplementary information.

5.3 Overhang Count, Symbol Size, and Redundancy Elimination

To understand how the entropy of the synthesized data may influence the choice of symbol size and overhang count when considering total reactions, Table 3 was constructed by varying the symbol size (in bits) and overhang count for both the ideal and *base-opt* case of removing redundancy. The values represent the geometric mean across all decompressed files in the Silesia corpus of the normalized reaction counts, where normalization is done with respect to the (1 bit, 3 overhangs) case. In Section 5.1, we showed that when applying no redundancy optimization, that although there are diminishing returns with larger overhang counts and symbol sizes, the total number of

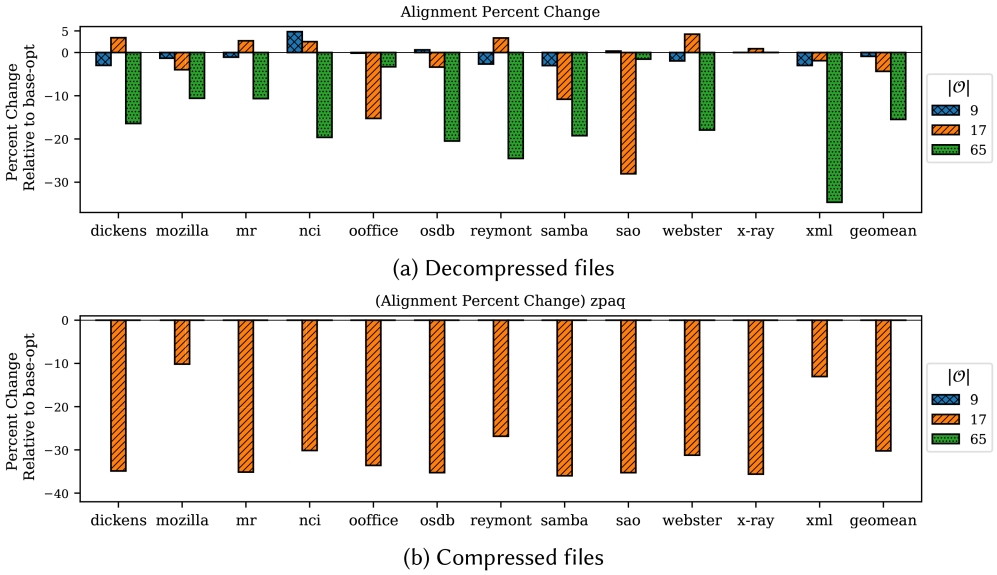


Fig. 8. Percent increase/decrease in reactions relative to *base-opt* when applying the *NOP* reaction padding for both decompressed and compressed files.

reactions decreases monotonically. In contrast, Table 3 shows that when optimization is factored in, the reaction count can actually increase when increasing both overhang count and symbol size. For example, *ideal* experiences a 1.64× increase and *base-opt* sees a 1.86× increase in reactions when going from (1 bit, 17 overhangs) to (1 bit, 65 overhangs). Both optimizations also see a 1.37× increase in reactions when increasing symbol size from (1 bit, 3 overhangs) to (3 bit, 3 overhangs). While increasing both parameters theoretically decreases tree size as shown in Section 5.1, the tree constructed with larger symbol/overhang set sizes will have larger data granularities at the same corresponding levels than that of smaller overhang/symbol sizes. If the smaller granularity facilitates enough redundancy elimination, then the benefits of a larger number of symbols or overhangs in terms of the number of reactions are lost due to a lack of redundancy. We find similar patterns in compressed files, but the results are omitted to save space.

5.4 Alignment Optimization

Figure 8 shows the percent change in reactions relative to *base-opt* when applying the alignment technique (Section 4.2) with the same parameters as Section 5.2. A negative percent change indicates the additional percentage of reactions removed over *base-opt* (better), and percent increases indicate that reactions are added (worse). These results show that adding padding reactions can decrease the reaction count for both compressed and decompressed files. Overhang set sizes of 3 and 5 are not present, since they are not large enough to satisfy the $|O-1|/4$ distance stipulation.

On average for decompressed files, we were able to reduce reaction counts by 0.88%, 4.35%, and 15.48% relative to *base-opt* for 9, 17, and 65 overhangs, respectively. Large benefits were observed for files where the addition of overhangs distributes the redundancy found by *ideal* over many infrequently used information states in *base-opt*. For example, *sao* experiences the largest decrease (28%) for 17 overhangs due to its larger information set that had less redundancy before alignment. For decompressed files, when $|O| = 65$ every file except *x-ray* sees decreases in reactions, since

redundancy is limited at this larger granularity in the *ideal* case, and distributing across many overhangs leads to many opportunities that fit the requirements for padding.

As discussed in Section 5.2, *zpaq* tends to increase the information set size for granularities like 16 bits, with the redundancy for each state being similar. Small data redundancy results in many unique states when considering overhangs, leading to many low-cost opportunities for padding. This results in a 30% decrease on average for $|O| = 17$.

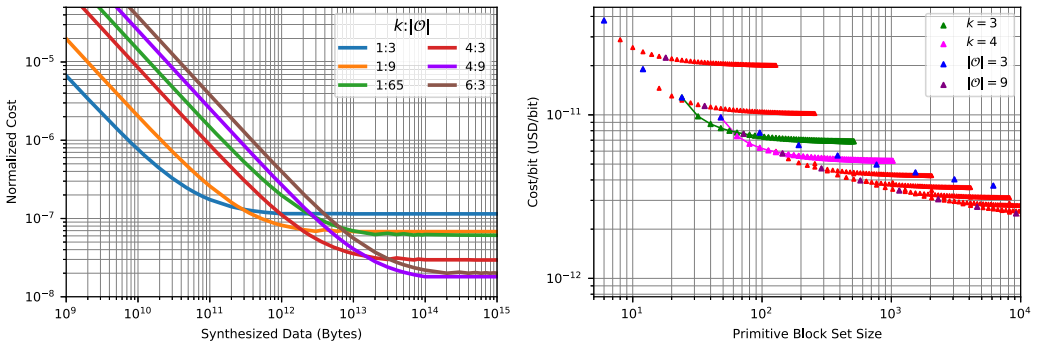
As mentioned in Section 4.2, there is indeed a tradeoff when improving data alignment to facilitate more sharing. In our implementation, we assume a system where both rotations of overhangs in the tree are synthesized with NOP codewords. For the 1-bit symbol sizes studied here, this leads to an approximate increase of 2× in the number of primitive blocks needed. A similar set size increase would be to increase k from 1-bit to 2-bits. However, Table 3 shows us that increasing the set size this way could in fact lead to more reactions for $|O| = 17$. The second tradeoff is in density, and so we measure the number of codewords inserted per strand on average for each benchmark of Figure 8(a) (decompressed data). The resulting geometric mean for the number of NOP codewords added per strand across the benchmarks for the three overhang settings $|O| = 9, 17, 65$ is 2.00, 7.49, 40.74, respectively. Assuming NOP codewords are the same length as symbol codewords, this leads to just a 0.9% additional codeword overhead at most considering the worst-case average of 40.74 codewords/strand and 1-bit symbols.

5.5 Cost Analysis

Our analysis up to this point is independent of the underlying chemical implementation of the assembly process. To further motivate the benefits of DINOS, it is necessary to pick an implementation to understand the approach's cost implications. For DINOS, there are two costs: costs of chemistries to assemble strands and costs to acquire the primitive blocks. For the former, we assume costs associated with a ligase enzyme that can be used to permanently connect together primitive blocks that have a structure like those blocks in Figure 2, and for the latter, we assume a *column-based de novo* synthesis approach. We compare this to using a *array-based de novo* synthesis approach for a whole set of data. An overview of these synthesis approaches, and why we choose these, can be found in Appendix Section A.3.1.

Figure 9(a) compares the cost of using DINOS to the cost of using *de novo* for the synthesis of the entire dataset. Each curve shows a relative cost for a select primitive block composition. Details about assumptions, the equations, and parameters used to generate these curves can be found in the Appendix Section A.3.2. The purpose of this cost model is not to consider detailed costs that would be incurred for developing a platform capable of large-scale data synthesis, but to rather compare the raw material cost incurred for constructing the DNA strands themselves. To be consistent with the evaluation analysis in the previous section, the strand length is assumed to be 4,096 bits. We assume *de novo* synthesis scales at a fixed cost/base in spite of low yields at such long lengths. Also note that we do not assume any cost reductions for coalescing reactions in this analysis, which may substantially lower the cost of DINOS even more.

There are three main costs associated with the materials required to construct DNA strands that encode information using assembly. One is that there is a cost associated with the procurement of the primitive block set. In this analysis, we assume that *de novo* synthesis is used to synthesize the single-stranded DNA needed to construct the single-double-single strand structure depicted in Figure 2. Therefore two unique strands must be synthesized per primitive block. The second cost of assembly is replenishing the primitive block set once all strands initially synthesized are used during assembly. We assume primitive block usage is evenly distributed across the set. Last, the third cost is that of materials such as enzymes and buffers that are needed to implement assembly.



(a) (b)

Fig. 9. (a) Cost to synthesize some amount of data using a code-word-based approach using ligation chemistry to implement assembly normalized to the cost of synthesizing the same amount of data using industry cost rates for *de novo* chemistry approaches. (b) Estimated USD/bit for various primitive strand set sizes and compositions. Each point represents the cost-per-bit for a unique number of symbols and overhangs, and the x-axis indicates the configuration's total primitive block size. As indicated in the legend, markers related to $k = 4$ represent the costs of primitive block sets that have a constant symbol size of 4 bits with varying overhang set sizes, likewise, $|O|$ markers are the set costs with constant overhang set sizes.

It is assumed that such materials are dependent on both the height of the assembly tree, and the number of DNA fragments that must be connected together in a single reaction.

When comparing these three costs against *de novo* synthesis for a volume of synthesized data, we can see from Figure 9(a), for the entire range (10^9B to 10^{15}B), that DINOS based on ligation chemistry is at least five orders of magnitude less than *de novo* synthesis. For small dataset sizes, the larger primitive block sets incur higher costs than that of smaller sets. This is because the dataset is not large enough to leverage the larger set of primitive blocks. However, this relationship generally inverts as the dataset increases. This is mainly due to reaction costs dominating the overall cost, because we estimate that each code word pool can be utilized on the order of 10^{12} times (Appendix A.3). This shows that it is cost-effective, from a consumable point of view, to decrease reactions by utilizing larger primitive block sets.

Another important trend is that, as dataset sizes increase for each configuration, the cost savings of using assembly improve relative to *de novo* and then asymptotically approach a minimum normalized cost. This trend occurs because at smaller dataset sizes, the cost of creating the initial set of strands is large compared to the cost of reactions and the cost of refilling the initial set. As the dataset becomes larger, this initial cost becomes negligible and the cost of assembly becomes dependent on two terms that are both proportional to the dataset size, resulting in the ratio of assembly costs to that of *de novo* costs to reach a constant value.

Our analysis concludes with a more detailed treatment of the choice of the composition of the primitive strand set. As mentioned previously, as the dataset to be synthesized approaches very large volumes, the larger primitive set typically results in lower costs. However, Figure 9(a) shows a contradiction to this statement. This can be seen by comparing the costs between the configurations ($k = 1, |O| = 65$), which has 130 primitive blocks, and ($k = 4, |O| = 3$), which has 48 blocks. Although the former has more than 2.7× more primitive blocks, at large dataset sizes it is about a factor of 2 more expensive. The reason for this is that increasing overhangs mainly affects cost by reducing the amount of materials by decreasing the height of the assembly tree. Since the height of an assembly tree is rapidly reduced as overhang set size increases from the minimum

of $|O| = 3$, cost savings also initially rapidly reduce. However, eventually increasing overhang set size has small impacts on height, thus increasing primitive block set sizes with minimal benefits. This indicates that choosing a configuration that minimizes cost for some desired primitive block set size requires carefully balancing both the number of code words and overhangs.

To better illustrate the impact of set construction on cost, Figure 9(b) shows the cost per bit against different primitive block set sizes for various combinations of code word and overhang set sizes. When keeping the symbol size constant and increasing overhangs for the curve shown by $k = 3$, we see that there are large cost decreases initially when increasing $|O|$. Eventually, as the cost benefits are exhausted from increasing the overhang set size for this symbol size, a curve corresponding to $k = 4$ appears directly beneath the curve of $k = 3$. This implies that increasing overhangs to reduce cost for $k = 3$ is no longer the optimal decision for increasing the primitive block set and thus the code word set should be expanded. However, eventually there are also diminishing returns for increasing the symbol size, since the number of leaf reactions are inversely related to the symbol size. This causes the curves to group closer in terms of cost, requiring large increases in the primitive block set size to gain relatively small cost benefits. For example, between 6 and 1,000 primitive blocks, costs can be decreased by a factor of 10.7. However, from 1,000 to 10^6 primitive blocks, the cost is only reduced by a factor of 2.2. This implies that most of the cost benefits can be realized with a relatively small number of primitive blocks.

5.6 Density Analysis

Since overhangs do not carry information in DINOS, they will negatively impact the density of the storage system. Table 4 provides a comparison for different symbol sizes on the density of DINOS compared to a standard short-strand that does not include overhangs. For our density analysis, we assume strands that encode 4,096 bits where each symbol is encoded to DNA using a base 3 representation[10, 27], 4 base overhangs [22, 30, 36, 37], and a total primer length of 40 bases [10, 27, 40]. For Short *de novo* synthesis, we keep the same encoding and primer overhead and assume a maximum strand length of 200 bases [9], but there are no overhangs. Formulas for calculating these densities are given in Section A.2. The table shows that for a binary code the overhangs have a significant effect on density, however, the gap diminishes as the symbol size is scaled up to a 26-bit code. The density of DINOS surpasses a short strand at 28 bits, since Short *de novo* is ultimately limited in density by the large fraction of bases devoted to its primers. However, if *de novo* synthesis progresses such that error rates are small in long strands, then the encoding density will be largely determined by the conversion from binary symbols to DNA. This results in Long *de novo* being more dense than assembly and is shown in the last row of Table 4. However, achieving strand lengths past 200 bases is difficult. Even with current *de novo* efficiencies being 99.5%, the yield rate is approximately just 36% for 200 base strands [9]. So, while the overhang sequences do add overhead, DINOS can have competitive density with *de novo* synthesis.

Now, we compare our assembly algorithms with existing techniques to better understand how efficiently overhangs are used from both a storage density and an algorithmic time and space complexity standpoint. We compare against Metclo and Short *de novo* oligo synthesis. We choose Metclo's assembly algorithm [22] as a comparison point, since they provide an algorithm that will work for any $|O|$, and the overhang overhead inserted into strands is identical to many other Golden Gate gene assembly methods such as Goldenbraid 2.0, Moclo, Loop Assembly, and so on [6, 29, 36, 37, 43]. Metclo is able to assemble $|O| - 1$ oligos in each reaction, and hence achieves identical time and space complexity to our work. However, it incurs more overhead from overhangs, requiring three overhangs between code words that are joined outside of leaf reactions. We give an in-depth treatment of Metclo in Appendix A.1.1.

Table 4. Raw Density Values in Bits/base for DINOS Compared to *de novo* and Metclo Assembly

Density (bits/base) Comparisons

	Symbol Size (bits)											
Synthesis Alg.	1	4	8	12	14	16	18	20	22	24	26	28
DINOS Density	0.2	0.57	0.79	0.99	1.06	1.05	1.11	1.16	1.2	1.18	1.22	1.25
Short *de novo*	0.8	1.07	1.07	1.2	1.24	1.16	1.2	1.23	1.26	1.2	1.22	1.24
Metclo $\|O\| = 3$	0.11	0.36	0.57	0.74	0.82	0.84	0.89	0.94	0.99	0.99	1.03	1.06
Metclo $\|O\| = 5$	0.14	0.44	0.66	0.85	0.92	0.93	0.99	1.04	1.08	1.08	1.11	1.15
Long *de novo*	1	1.33	1.33	1.5	1.56	1.45	1.5	1.54	1.57	1.5	1.53	1.56

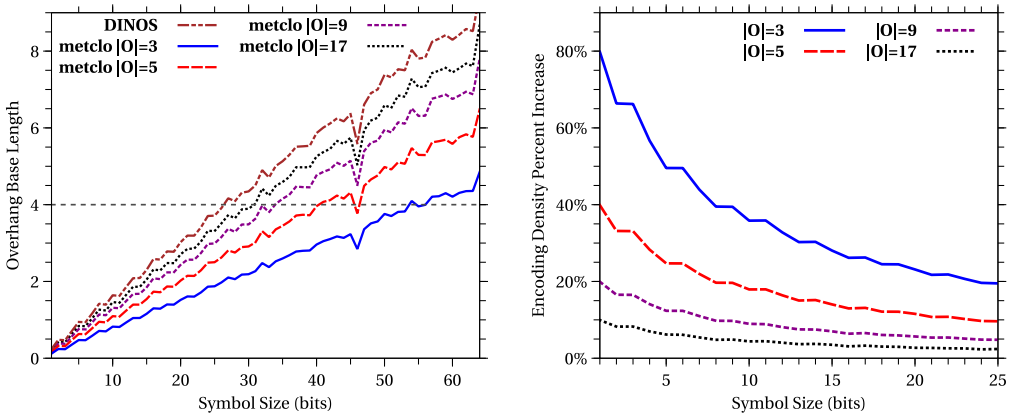


Fig. 10. Left: Curves that outline operating points in which assembly is as dense as *de novo* synthesis. Gray line represents a lower bound, and each curve for each assembly type is an upper bound, both bounds inclusive. Right: Percent increase in storage density when using our approach compared to Metclo for different $|O|$.

To compare density of our assembly method with Metclo, we calculate the minimum symbol size so density is equivalent to the density of Short *de novo* synthesis. Using that same analysis, we calculate the percent increase in storage density compared to Metclo that our work provides. We do this by taking a strand designed for each system, counting the bases that contribute to overhead, and then calculating the resulting density.

The left chart of Figure 10 solves for the upper bound of overhang length that still allows for equivalent density to *de novo* synthesis for assembly approaches. A line is drawn at the 4 base marker. Any symbol size and overhang length combinations that fall in between the lower bound and the upper bound (inclusive) indicate that the assembly approach will have an equal or higher storage density than *de novo* synthesis. We see that our approach intersects the lower bound at 27 bits compared to 31 bits for Metclo configured with $|O| = 17$. Note, our system results in the same density regardless of the number of overhangs, but for Metclo, increasing overhang set size reduces the number of boundaries in which there are three overhangs between code words resulting in the upper bound curves shifting to the left. This implies that even if Metclo and our approach had coinciding intersection points, Metclo would require a larger primitive block set, making it less desirable.

The chart on the right of Figure 10 normalizes the storage density of our algorithm to Metclo. Since symbol size and overhang set size decrease the number of times in which 3 overhangs are

inserted between code words, the gap between the density of Metclo and our approach decreases with larger numbers of symbols. At the point of the smallest primitive block set size we support (1 bit, 3 overhangs), our approach provides 80% greater storage density. Closing the gap between storage densities requires Metclo to grow their primitive block set. For example, if Metclo uses 17 overhangs and our approach uses 3, for 5.3× more strands ((17-1)/3), Metclo can reduce the gap to 10% at 25-bit symbols. Raw densities are also included for Metclo in Table 4 for $|O| = 3, 5$ and a subset of symbol sizes shown in Figure 10.

6 PRACTICAL CONSIDERATIONS AND FUTURE WORK

A demonstration of DINOS in a wet lab involves many practical considerations and is part of our future work. An important consideration is choosing the DNA sequences for the overhangs and codewords. For the case of overhangs, there has been an extensive analysis by Potapov et al. [30] to profile all 256^2 pairs of 4 base pair overhangs to find high-fidelity overhang sets that minimize unwanted bindings between different overhangs. They provide a set of 20 overhangs with high fidelity that can cover all configurations we studied except for $|O| = 65$. Extending the set to this larger size would require screening longer overhangs for high fidelity and choosing a set that works well together.

The design of the codeword sequences is also important and can influence the fidelity of assembly and successful decoding of data. Assuming that a single-double-single strand complex is to be used as the primitive block, as shown in Figure 2, each side of the complex may be synthesized individually. Then, using hybridization, the complex can be constructed. To achieve high-fidelity primitive blocks, code words should be designed such that there are no secondary structures that may disrupt the alignment of the two sides of the block. Furthermore, it may be advantageous if codewords are also designed with high Hamming distances so synthesis errors can be tolerated and for further limiting non-specific interactions [27, 40, 41]. Finally, the double-strand region should have a high enough melting temperature to avoid dissociation at the temperatures needed for ligation-reaction protocols [30].

In addition to designing sequences for codewords and overhangs, incomplete assemblies can pose a problem during assembly reactions. Unreacted products with exposed overhangs can propagate up the tree and assemble with other unreacted products to form undesired data strings. Similar problems have been documented in a DNA implementation of a stack data structure, where incomplete products of each stage interfered with products of a later stage [24]. Given that reactions generally do not reach 100% completion, strategies for purifying products, improving the yield rates, or even error correction approaches for dealing with incorrect products need further study.

7 RELATED WORK

The applications of DNA self-assembly and redundancy elimination in storage and computer systems are both well studied. Like our work, DNA self-assembly works rely on a small set of regular DNA structures, referred to as tiles, that generate larger structures. These tiles can be used to generate grids with predictable patterns that act as scaffolds to implement highly conductive nano-wires or to use as the groundwork for assembling arrays of nanoparticles [28, 38, 46]. Tiles have also been shown to self-assemble 3-D structures rather than just 2-D grids [18].

The self assembly of DNA tiles has also been studied extensively as a means to achieve universal computation [44]. Interest in DNA-tile computing has led to self-assembly implementations of binary counters and cellular automatons such as the XOR cellular automaton implementation that builds DNA crystals that represent Sierpinski Triangles [8, 35]. Furthermore, DNA-tile-computation is able to solve NP-complete problems, such as the Hamiltonian path

problem, by verifying the result of many possible solutions in parallel with just a polynomial number of unique tiles that represent paths in the graph [5, 44].

Eliminating redundancy is a common optimization in storage and computer systems. For storing information, deduplication that eliminates duplicates of large chunks of data within and across files has been shown as an effective method to reduce the storage needed for backup and archival storage systems [15, 45]. Compression techniques like dictionary based compression that maintain the full integrity of the data, or lossy techniques like JPEG that eliminate low-importance information, can also be used with deduplication or on their own to further reduce the amount of storage needed [25, 42, 47].

In computing systems, removing redundancy typically means to avoid repeating instructions when the results of those instructions are already available. This can be done at the compiler level or by using hardware caches to save results of functions that are computed redundantly [13, 21, 33]. Most analogous to our approach is common sub-expression elimination, a technique employed by compilers to remove redundant instructions [21]. Common sub-expression elimination identifies two or more instructions with the same opcode and same operands and eliminates all but one, a process that inspired how we consolidate redundant reactions. However, while the optimizations are similar in principle, building strands of information from DNA blocks require new representations and optimizations. Existing compilers and tools were not sufficient for this purpose.

8 CONCLUSION

The DINOS framework is designed to facilitate the synthesis of data strands and makes it possible to synthesize arbitrary strands of data from a small set of code words and a set of overhangs using only $|O| \cdot 2^k$ primitive blocks. The tunable nature of this system allows it to operate with as few as six primitive blocks. Furthermore, our analysis suggests that a balance between the number of overhangs and the number of input symbols must be taken into account to achieve the best results for information density, the number of reactions, and the cost of the system. Furthermore, the cyclic overhang assignment method is simple and versatile, making it easy to reason about assembly. Configured with only six primitive blocks (e.g., a binary code with three overhangs), DINOS makes it feasible and affordable to assemble a small library of strands manually.

DINOS also facilitates larger granularity redundancy elimination by leveraging analysis of the assembly tree representation. A redundant reaction in the assembly tree may coalesce with any other equivalent product both within and across strands. Using our approach, we were able to reduce reactions by an average of 91% and 41.2%, for decompressed and compressed files, respectively, relative to no optimization. By applying an alignment technique to find even more redundancy, reactions can be reduced further to an average of 91.1% and 59% for decompressed and compressed data, respectively. The benefits of these approaches are significant and surprising. Using these optimizations, we found that synthesizing the decompressed version of a file can reduce reactions by 60% compared to its compressed version.

We also compare DINOS to state-of-the-art synthesis and assembly approaches. We show that it offers competitive information density over a range of primitive block designs. Using our cost model, we demonstrate the potential to substantially lower synthesis costs compared to *de novo* synthesis.

For future work, we plan to fully automate and tune DINOS for the assembly of large volumes of data. In so doing, we will further tune the assembly process for data and optimize the chemistry and DNA sequences that work well with the design. This will also serve to refine and inform our comparison to *de novo* synthesis and other assembly processes. We hope DINOS encourages new research for data-inspired synthesis and hope it encourages new approaches for analyzing and exploiting the inherent redundancy in the data-strand assembly process.

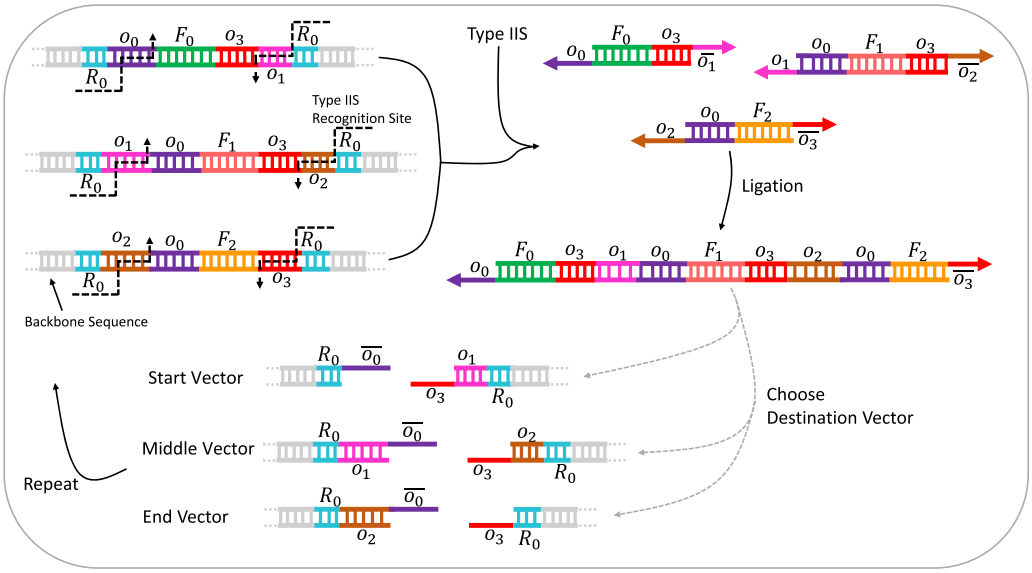


Fig. 11. Metclo hierarchical assembly. Three information fragments F_0, F_1, F_2 assembled from a previous step are freed from a backbone strand with the use of a Type IIS restriction enzyme that has recognition site R_0. The enzyme cuts outside of the site causing the release of each fragment with a pair of overhangs oriented such that the three fragments can assemble in a ligation step. The resulting assembly is encased in the appropriate destination vector such that the process can be repeated.

A APPENDIX

A.1 Metclo Analysis

A.1.1 Metclo Description. Figure 11 provides a small example of Metclo where $|O| = 4$, though there will be no loss of generality. Assembly in Figure 11 starts with three information fragments F_0, F_1, F_2 that are assumed to be assembled in a previous step. The fragments are housed in a backbone strand and are released through Type IIS restriction enzyme digestion that cuts outside the restriction site R_0. The restriction site is oriented to ensure that overhangs are produced in a manner that allows fragments to assemble in a subsequent assembly and ligation step. After assembly, the final step is to choose a destination vector. There are three choices, since $|O| = 4$, a start, middle, and end vector. These vectors determine the positioning of the new strand in the next assembly reaction. After placing the result in a vector, the process can be repeated again until the desired strand length is reached. When the overhang set is increased, the only change is that the set of middle vectors will expand to accommodate the $|O| - 3$ possible middle positions in an assembly. Note, since each assembly will have the same bookend overhangs, vectors are required to ensure unique overhangs in subsequent reactions, leading to three overhangs between code words that are joined together after the initial leaf reactions.

A.1.2 Metclo Primitive Strand Set Analysis. To give a full comparison with Metclo, and to view their system from the perspective of information storage, we also analyze the primitive strand set size, $|\mathcal{U}_{Metclo}|$, for Metclo. Since Metclo wraps the start and end overhang each reaction, Metclo does not need strands of the form $o_{|O|-1} \cdot c \cdot \overline{o_0}\ \forall c \in C$, which we use to complete the cycle of overhangs to avoid more than one overhang between code words. In other words, Metclo only requires a version of each code word that corresponds to the start, middle, and end vectors. Since

there are only $|O| - 1$ vectors, the size of Metclo's primitive strand set becomes $|\mathcal{U}_{Metclo}| = 2^k \cdot (|O| - 1)$.

A.2 Detailed Density Analysis

The number of bases inserted by overhangs for our system, B_o, can be calculated as:

$$B_o = L_o + \left\lceil \frac{I_{ha}}{k} \right\rceil \cdot L_o, \tag{1}$$

where L_o is the length of overhangs in bases, I_{ha} is the length in bits of a strand that can be assembled by hierarchical assembly, and k is symbol size in bits. This expression counts the number of overhang bases on the right of each code word plus the overhang bases on the left of the first code word in the strand. For Metclo, an additional term is required to factor in the cases in which there are three overhangs between code words. This will occur at every boundary between two sub-assemblies of size $k \cdot (|O| - 1)$ bits, and the total overhang bases for Metclo, $B_{o,Metclo}$ can be expressed as the following:

$$B_{o,Metclo} = L_o + \left\lceil \frac{I_{ha}}{k} \right\rceil \cdot L_o + \left(\left\lceil \frac{I_{ha}}{k \cdot (|O| - 1)} \right\rceil - 1 \right) \cdot 2 \cdot L_o. \tag{2}$$

The first two terms calculate the need for at least one overhang in between codewords, and the third term accounts for the two extra overhangs between code words that are not assembled in the leaves. With the number of overhang bases inserted into the final strand known for both approaches, we can now calculate the density (bits/base) of a hierarchical assembly system, D_{ha}. Generically, this can be described with the following equation:

$$D_{ha} = \frac{I_{ha}}{B_o + L_p + \lceil I_{ha}/k \rceil \cdot \lceil k/\sigma_{ha} \rceil}, \tag{3}$$

where L_p is the number of bases allocated in the strand for primers, σ_{ha} is the raw encoding density for the encoding process that converts payload information into bases for hierarchical assembly synthesis, B_o is the total number of overhand bases in a strand constructed with assembly, k is the length in bits for a symbol. Finally, we calculate the density of a *de novo* synthesis system to be able to calculate operating points in which assembly is equally dense. The density of *de novo* synthesis, D_{dn}, can be calculated as:

$$D_{dn} = \frac{(B_{dn} - L_p) \cdot \sigma_{dn}}{B_{dn}}, \tag{4}$$

where σ_{dn} is the raw encoding density for the process that converts payload information into bases for *de novo* synthesis, and B_{dn} is the length (in bases) of a strand that can be grown by *de novo* synthesis. With the densities of the two synthesis approaches known, we solve the inequality of Equation (5) for L_o for both our assembly system and Metclo. This provides an upper bound, as shown in Figure 10, for overhang length that is allowable for the density of assembly synthesis to be equivalent to *de novo* synthesis.

$$\frac{(B_{dn} - L_p) \cdot \sigma_{dn}}{B_{dn}} \leq \frac{I_{ha}}{B_o + L_p + \lceil I_{ha}/k \rceil \cdot \lceil k/\sigma_{ha} \rceil} \tag{5}$$

A.3 Cost Analysis

A.3.1 Principles of DNA Synthesis Costs. Here, we provide an overview of *de novo* and assembly synthesis along with quantitative insight into why assembly can be a much cheaper synthesis alternative for data. Figure 12 shows a comparative overview of standard *de novo* synthesis on the left and the general idea of assembly on the right. *De novo* synthesis grows each strand that

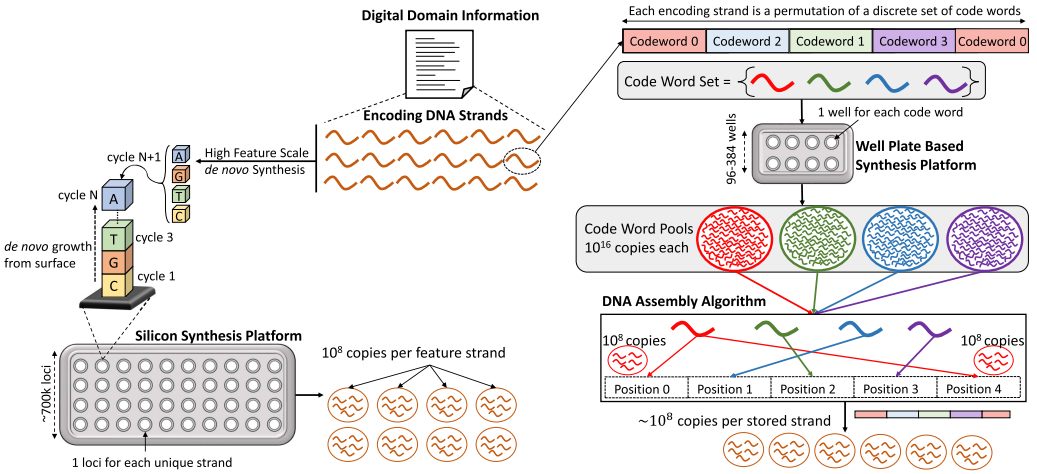


Fig. 12. Comparison of an assembly synthesis flow for constructing the encoding strands of a digital information source against a synthesis flow that relies completely on large feature scale *de novo* synthesis. Both systems use *de novo* synthesis, but in different ways. Large feature scale synthesis relies on decreasing synthesis costs by sacrificing strand copy yield for the number of unique strands generated per synthesis run. Assembly synthesis leverages the large copy scale afforded by low feature scale *de novo* synthesis to attain a per-code-word instantiation cost that is several orders of magnitude smaller.

encodes data one base at a time, and each unique encoding strand gets a specific location to grow on a silicon synthesis platform. The main mechanism that has been used to scale down *de novo* synthesis costs has been to scale the platform in which the oligos are built. For example, Twist Bioscience utilizes a silicon substrate technology to allow for up to 700K unique oligos to be built on a single chip with a cost of $4.4 \cdot 10^{-4}$ USD/base [2]. This category of *de novo* synthesis is known as *array-based synthesis*. Another well-known *de novo* synthesis method, termed as *column synthesis*, focuses on higher-quality strands at a lower unique-strand scale [20]. Because column synthesis generates unique strands on 96/384 well plates, the cost per unique base can be three orders of magnitude higher than array-based synthesis at 0.29 USD/base [1].

On the right side of Figure 12, we have the general flow of an assembly-based synthesis process. Here, a strand of data to be synthesized is broken down into five codeword regions, where four are unique. We can now just synthesize this small set of codewords and assemble them using ligation chemistries that use a ligase enzyme to fully connect the backbone of the overhang with the backbone of the neighboring strand [22, 30, 37]. From this, we recognize that we really do not need many unique features for such a small set of codewords. Furthermore, column-based synthesis is in fact more cost-effective than array-based synthesis if many copies of a single feature are desired. For example, array-based synthesis provides 10^8 copies, while column synthesis can provide copy scales of 10^{16} at the same previously mentioned cost per base for both [1, 2]. This leads to column-based synthesis being five orders of magnitude cheaper at some copy scale. Thus, if column synthesis is paired with assembly-based synthesis to form longer strands with meaningful amounts of data, data synthesis costs can potentially be improved greatly.

A.3.2 Calculating Costs. This section supplements the figures presented for cost analysis in Section 5.5 with equations used to create those figures. We end this section with set of parameters used to implement the figures from the equations. The cost of assembly synthesis, as mentioned before, takes into account the cost of obtaining the primitive block set with *de novo* synthesis,

maintenance of that set, and the materials expended on the reactions within the tree. These costs are summarized by the following equation:

$$C_{assembly} = \left\lceil \frac{T_{bits}}{k \cdot (|O| - 1)} \right\rceil \cdot C_{LR} + (\alpha \cdot 2^k \cdot |O|) \cdot \left(C_{dn} \cdot \left(\frac{k}{\sigma_{ha}} + L_o \right) \right) \cdot \left\lceil \frac{T_{bits}}{2^k \cdot k \cdot |O| \cdot N_{uses}} \right\rceil. \quad (6)$$

The first term in the equation represents the cost associated with reactions needed to assemble strands, where C_{LR} is used to represent the cost of leaf reactions and as we will show later includes the costs of intermediate nodes in the tree. The second term calculates the cost associated with acquiring the first set of primitive blocks, along with the cost of replenishing that set as data is synthesized. In this equation, T_{bits} is the total number of bits synthesized, α is a multiplier used to model the number of DNA strands needed per primitive block, C_{dn} represents the per-base cost associated with the *de novo* synthesis process used for instantiating the initial set of primitive blocks, and N_{uses} represents the number of times a primitive block can be reused after being obtained from *de novo* synthesis. The cost of leaf reactions is modeled by using scaling factors applied to a cost-per-overhang rate R_{LR}. The first scaling factor, $|O| - 1$, is used to model the increase in materials needed to support more ligation in a single reaction. The second term is used to model the increase in materials required to support deeper trees. The model assumes that each reaction in a tree requires the same amount of ligation materials as a single leaf reaction. However, we can work this cost into the cost of each leaf reaction by spreading out the cost of an internal node amongst its children, and from there recursively until the leaf is reached. Thus, the additional cost for an internal node will be split to a leaf reaction needed by some power of $|O| - 1$, depending on the internal node's height. This is shown below in Equation (7).

$$C_{LR} = R_{LR} \cdot (|O| - 1) \cdot \left(1 + \sum_{i=1}^{h-1} \frac{1}{(|O| - 1)^i} \right) \quad (7)$$

$$h = \log_{|O|-1} \left(\frac{I_{ha}}{k} \right)$$

With the cost of leaf reactions defined, Equation (6) is normalized by the simple factor $C_{dn,File} \cdot \frac{T_{bits}}{\sigma_{dn}}$ to produce the curves of Figure 9(a). $C_{dn,File}$ represents the cost per base of the *de novo* synthesis process used to synthesize all strands for a file. To create the points in Figure 9(b) that show the cost per bit for various primitive block set sizes, it is assumed that the amount of data is asymptotically large, e.g., $T_{bits} \rightarrow \infty$. This allows the ceiling functions to be dropped, since the cost impact of moving the divisions to the next greatest integers for large T_{bits} is negligible. This allows T_{bits} to be factored out from both terms that represent reaction costs and primitive set synthesis costs, leading to the remaining factor being the cost per bit plotted in Figure 9(b).

To generate Figures 9(a) and 9(b), we assume that the costs of the assembly approach are purely the cost of the consumable materials, e.g., the cost of ligase enzyme used to implement the connection of primitive blocks and the oligos ordered to fill the primitive block set. So, we do not consider the costs of personnel or liquid handling platforms that would be needed to physically implement the algorithm at a sufficient throughput. As our cost comparison reference that represents the cost of *de novo* synthesis for an entire file's strands, we choose Twist Bioscience's prices, since their synthesis process is tailored towards optimizing the cost per base for each unique ordered base [2]. To take into account economy of scale, we assume Twist Bioscience's lowest cost per base of $4.4 \cdot 10^{-4}$ USD/base when ordering 250 base pair strands on the largest plate of 696,000 unique oligos at a price of 76,560 USD. Although using this cost is a consumer-facing value that hides the actual costs of the raw consumables, since a company must factor in profit, personnel, and so on, the values we use to calculate the cost of assembly are also consumer-facing, providing

a fair comparison. Comparing costs of synthesis is done in a similar manner for other emerging synthesis techniques for DNA data storage as described by Reference [7].

For *de novo* synthesis, costs attributed to building the primitive block set for assembly are chosen with scale of synthesis yield being a priority. Thus, we choose Eurofins custom oligos, which can be ordered at a rate of 0.29 USD per ordered base pair at a synthesis yield of 50 nmol, which, through our experience, yields approximately 25 nmol of strands for strand lengths of 17 base pairs, which is similar to all strand lengths considered in Figures 9(a) and 9(b) [1]. Such a yield at this cost per base pair corresponds to a scale adjusted base pair cost of 0.0116 (USD/bp)/nmol, which is much more favorable for assembly synthesis than Twist Bioscience's process, which only guarantees 0.2 fmol yield corresponding to 2200 (USD/bp)/nmol [2].

For other consumables outside of oligos, we assume that the only remaining chemical costs are that for the DNA ligase enzyme, as this is the dominant cost factor of creating the mixture that facilitates overhang ligation. For this cost, we assume New England BioLabs' T4 DNA ligase [4]. The cost of this ligase is used to calculate the cost per leaf reaction per overhang R_{LR}. To do this, we calculate a cost per unit of T4 DNA ligase, e.g., $260/10^5$ USD/unit. A *unit* is defined as the amount required to ligate 50% of 0.12 μM of 5′ DNA termini in a reaction volume of 20 μl over 30 minutes at 16°C [4]. A termini can be defined as a physical overhang in the reaction that must be ligated. Using this information, we can calculate the fraction of a unit required for a single termini in the reaction with the following calculation:

$$\frac{\text{unit}}{\text{termini}} = \frac{1}{0.12 \cdot 10^{-6} \frac{\text{moles}}{\text{liters}} \times 20 \cdot 10^{-6} \text{ liters} \times 6.022 \cdot 10^{23} \frac{\text{overhangs}}{\text{mole}}} = 6.92 \cdot 10^{-13}. \tag{8}$$

Next, we need to know the number of overhangs that will be present in any given reaction. We assume that we need 10^4 copies of each primitive block in a leaf reaction. Thus, if the reactions were 100% efficient and converted each codeword copy into a complete strand, then there would be 10^4 copies per strand. Although, realistically, these reactions will not be 100% efficient, the number of copies required downstream in the storage system is several orders of magnitude smaller. As shown by Chen et al., as few as 10 copies per oligo can be accessed with typical random access methods such as **polymerase chain reaction (PCR)** [12]. Thus, our assumption holds as reasonable if a given reaction tree can be built with at least $10/10^4 = 0.1\%$ efficiency. With 10^4 copies per codeword, the number of total overhangs to be ligated will be $(|O|-1)\cdot 10^4$. Thus, we can calculate the amount of unit needed for each reaction per logical overhang as:

$$\frac{\text{unit}}{\text{logical overhang}} = 6.92 \cdot 10^{-13} \frac{\text{unit}}{\text{termini}} \times 10^4 \frac{\text{termini}}{\text{logical overhang}} = 6.92 \cdot 10^{-9}. \tag{9}$$

Finally, the cost per logical overhang in the leaf reaction (R_{LR}) can be calculated as

$$R_{LR} = 6.92 \cdot 10^{-9} \frac{\text{unit}}{\text{logical overhang}} \times \frac{260}{10^5} \frac{\text{USD}}{\text{unit}} = 1.79 \cdot 10^{-11} \frac{\text{USD}}{\text{logical overhang}}. \tag{10}$$

As for the remaining parameters, L_o was chosen to be 4 bases, as this is a typical overhang length for Golden Gate assembly [30], a base-3 encoding was assumed as in Section 5.6, which translates to a density of 1.33 bits per base, α was set to 2 because each primitive block needs 2 individual strands to be created from the initial *de novo* synthesis to construct blocks similar to those in Figure 2, and last, N_{uses} can be easily calculated from the 25 nmol synthesis yield and the number of code-word copies used each reaction to get $N_{uses} = 1.5 \cdot 10^{12}$.

A.4 Proof of Unique Tree Rotations

As pointed out in Section 4.2, if there are only two unique rotations through the overhang set O, then only $2|O|$ *NOP* strands are required to represent the result of any *NOP* tree root. The following

corollary proves this and provides the exact overhang rotations that exist at each level of the tree, allowing for *NOP* strands to be fully specified.

Corollary A.1. *Given the $|O|-1$ degree assembly tree constructed using a cyclic assignment of overhangs to generate the primitive blocks as described by Theorem 3.3, the generators for each level of the assembly tree generate exactly two unique permutations of the cyclic group $\mathbb{Z}_{|O|}$. Furthermore, let l be a given level of the tree counting from the leaves starting at 0, the permutation generated by even l is described by the sequence $a_n = n \; \forall \; n \in [0, |O|-1]$ and for odd l is described by $a_n = |O| - n \; \forall \; n \in [0, |O|-1]$. a_n is the nth generated value of $\mathbb{Z}_{|O|}$.*

Proof. First, consider the first two layers of the tree. Using a stride of 1 at the primitive block level ($l = 0$), we can see trivially that $\mathbb{Z}_{|O|}$ is generated in the order $a_n = n \; \forall \; n \in [0, |O|-1]$. For $l = 1$, the generating stride is $|O|-1$, and thus the nth generated value of the group can be expressed by $n \cdot (|O|-1) \pmod{|O|}$, where n is in the range $[0, |O|-1]$. We can write $n \cdot (|O|-1)$ as $|O| \cdot (n-1) + (|O| - n)$, wherein we can apply Euclid's Division Lemma to see that $(|O| - n) \equiv |O| \cdot (n-1) + (|O| - n) \pmod{|O|}$. Thus, for $l = 1$, $a_n = |O| - n$.

These two sequences for $l = 0$ and $l = 1$ are clearly different. Thus, we must show that these are the only two unique sequences for all $l > 1$. For odd $l > 1$, the stride becomes an l-degree polynomial of the form $(|O|-1)^l = \alpha_l |O|^l + \alpha_{l-1}|O|^{l-1} \ldots + l|O| - 1$. We can write this as $\alpha_l |O|^l + \alpha_{l-1}|O|^{l-1} \ldots + (l-1)|O| + |O| - 1 = |O| \cdot \mathbf{P}(|O|) + |O| - 1$, where $\mathbf{P}(|O|)$ is a polynomial of $|O|$ with degree $l-1$. By multiplying by n and rewriting like before for $l = 1$, we get $n \cdot |O| \cdot \mathbf{P}(|O|) + |O| \cdot (n-1) + |O| - n = |O| \cdot (n\mathbf{P}(|O|) + n - 1) + |O| - n \equiv |O| - n \pmod{|O|}$, which shows that for all odd l the sequence of overhangs is that of $l = 1$.

The same approach is taken for even $l > 1$, resulting in $\alpha_l |O|^l + \alpha_{l-1}|O|^{l-1} \ldots - l|O| + 1$. Notice the sign of the last term is now positive. Multiplying by n and simplifying to separate the portion that is a multiple of $|O|$, we get $n \cdot |O| \cdot \mathbf{P}(|O|) + n \equiv n \pmod{|O|}$. Thus, showing that for all even l the sequence of integers is the same of $l = 0$, and furthermore shows there are only two unique sequences of overhang rotations. □

REFERENCES

[1] [n.d.]. DNA/RNA Price List. Eurofin. Retrieved 22 Feb., 2021 from https://eurofinsgenomics.com/en/products/dnarna-synthesis/oligo-price-list.

[2] [n.d.]. Oligo Pools. Twist Bioscience. Retrieved 29 Aug., 2019 from https://www.twistbioscience.com/sites/default/files/resources/2019-09/ProductSheet_OligoPools_29Aug19_Rev5.1.pdf.

[3] Sebastian Deorowicz. Silesia Compression Corpus. Retrieved from http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia.

[4] [n.d.]. T4 DNA Ligase. New England BioLabs Inc. Retrieved 22 Feb., 2021 from https://www.neb.com/products/m0202-t4-dna-ligase#Product%20Information_Properties%20&%20Usage.

[5] Leonard M. Adleman. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266, 5187 (Nov. 1994), 1021–1024. DOI: https://doi.org/10.1126/science.7973651

[6] Andreas I. Andreou and Naomi Nakayama. 2018. Mobius assembly: A versatile golden-gate framework towards universal dna assembly. *PLoS One* 13, 1 (Jan. 2018), e0189892. DOI: https://doi.org/10.1371/journal.pone.0189892

[7] Philipp L. Antkowiak, Jory Lietard, Mohammad Zalbagi Darestani, Mark M. Somoza, Wendelin J. Stark, Reinhard Heckel, and Robert N. Grass. 2020. Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction. *Nat. Commun.* 11, 1 (Oct. 2020), 5345. DOI: https://doi.org/10.1038/s41467-020-19148-3

[8] Robert D. Barish, Paul W. K. Rothemund, and Erik Winfree. 2005. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Lett.* 5, 12 (Dec. 2005), 2586–2592. DOI: https://doi.org/10.1021/nl052038l

[9] Bryan Bishop, Nathan McCorkle, and Victor Zhirnov. 2017. Technology working group meeting on future DNA synthesis technologies. (Oct. 2017), 39. Semiconductor Research Corporation.

[10] James Bornholt, Randolph Lopez, Douglas M. Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. 2016. A DNA-based archival storage system. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. ACM, New York, NY, 637–649. DOI: https://doi.org/10.1145/2872362.2872397

[11] D. Carmean, L. Ceze, G. Seelig, K. Stewart, K. Strauss, and M. Willsey. 2019. DNA data storage and hybrid molecular–electronic computing. *Proc. IEEE* 107, 1 (Jan. 2019), 63–72. DOI: https://doi.org/10.1109/JPROC.2018.2875386

[12] Yuan-Jyue Chen, Christopher N. Takahashi, Lee Organick, Callista Bee, Siena Dumas Ang, Patrick Weiss, Bill Peck, Georg Seelig, Luis Ceze, and Karin Strauss. 2020. Quantifying molecular bias in DNA data storage. *Nat. Commun.* 11, 1 (June 2020), 3264. DOI: https://doi.org/10.1038/s41467-020-16958-3

[13] Cliff Click. 1995. Global code motion/global value numbering. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'95)*. Association for Computing Machinery, New York, NY, 246–257. DOI: https://doi.org/10.1145/207110.207154

[14] Yaniv Erlich and Dina Zielinski. 2017. DNA Fountain enables a robust and efficient storage architecture. *Science* 355, 6328 (Mar. 2017), 950–954. DOI: https://doi.org/10.1126/science.aaj2038

[15] Abdullah Gharaibeh, Cornel Constantinescu, Maohua Lu, Ramani Routray, Anurag Sharma, Prasenjit Sarkar, David Pease, and Matei Ripeanu. 2014. DedupT: Deduplication for tape systems. In *Proceedings of the 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–11. DOI: https://doi.org/10.1109/MSST.2014.6855555

[16] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. 2013. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* 494, 7435 (Feb. 2013), 77–80. DOI: https://doi.org/10.1038/nature11875

[17] Robert N. Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J. Stark. 2015. Robust chemical preservation of digital information on DNA in silica with error-correcting codes. *Angewandte Chemie Int. Ed.* 54, 8 (Feb. 2015), 2552–2555. DOI: https://doi.org/10.1002/anie.201411378

[18] Yu He, Tao Ye, Min Su, Chuan Zhang, Alexander E. Ribbe, Wen Jiang, and Chengde Mao. 2008. Hierarchical self-assembly of DNA into symmetric supramolecular polyhedra. *Nature* 452, 7184 (Mar. 2008), 198–201. DOI: https://doi.org/10.1038/nature06597

[19] Reinhard Heckel, Ilan Shomorony, Kannan Ramchandran, and David N. C. Tse. 2017. Fundamental limits of DNA storage systems. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*. 3130–3134. DOI: https://doi.org/10.1109/ISIT.2017.8007106

[20] Randall A. Hughes and Andrew D. Ellington. 2017. Synthetic DNA synthesis and assembly: Putting the synthetic in synthetic biology. *Cold Spring Harb. Perspect. Biol.* 9, 1 (Jan. 2017). DOI: https://doi.org/10.1101/cshperspect.a023812

[21] Aho Lam and Sethi Ullman. 2014. *Compilers Principles, Techniques, and Tools* (2nd ed.). Pearson Education, Edinburgh Gate, Harlow, Essex, England.

[22] Da Lin and Christopher A. O'Callaghan. 2018. MetClo: Methylase-assisted hierarchical DNA assembly using a single type IIS restriction enzyme. *Nucleic Acids Res.* 46, 19 (Nov. 2018), e113. DOI: https://doi.org/10.1093/nar/gky596

[23] Kevin N. Lin, Kevin Volkel, James M. Tuck, and Albert J. Keung. 2020. Dynamic and scalable DNA-based information storage. *Nat. Commun.* 11, 1 (June 2020), 2981. DOI: https://doi.org/10.1038/s41467-020-16797-2

[24] Annunziata Lopiccolo, Ben Shirt-Ediss, Emanuela Torelli, Abimbola Feyisara Adedeji Olulana, Matteo Castronovo, Harold Fellermann, and Natalio Krasnogor. 2021. A last-in first-out stack data structure implemented in DNA. *Nat. Commun.* 12, 1 (Aug. 2021), 4861. DOI: https://doi.org/10.1038/s41467-021-25023-6

[25] Matt Mahoney. 2016. ZPAQ. Retrieved from http://mattmahoney.net/dc/zpaq.html.

[26] Matt Mahoney. 2019. Silesia Open Source Compression Benchmark. Retrieved from http://mattmahoney.net/dc/silesia.html.

[27] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z. Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher N. Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze, and Karin Strauss. 2018. Random access in large-scale DNA data storage. *Nat. Biotechnol.* 36, 3 (Mar. 2018), 242–248. DOI: https://doi.org/10.1038/nbt.4079

[28] Sung Ha Park, Constantin Pistol, Sang Jung Ahn, John H. Reif, Alvin R. Lebeck, Chris Dwyer, and Thomas H. LaBean. 2006. Finite-size, fully addressable DNA tile lattices formed by hierarchical assembly procedures. *Angewandte Chemie Int. Ed.* 45, 5 (2006), 735–739. DOI: https://doi.org/10.1002/anie.200503797

[29] Bernardo Pollak, Ariel Cerda, Mihails Delmans, Simón Álamos, Tomás Moyano, Anthony West, Rodrigo A. Gutiérrez, Nicola J. Patron, Fernán Federici, and Jim Haseloff. 2019. Loop assembly: A simple and open system for recursive fabrication of DNA circuits. *New Phytol.* 222, 1 (2019), 628–640. DOI: https://doi.org/10.1111/nph.15625

[30] Vladimir Potapov, Jennifer L. Ong, Rebecca B. Kucera, Bradley W. Langhorst, Katharina Bilotti, John M. Pryor, Eric J. Cantor, Barry Canton, Thomas F. Knight, Thomas C. Evans, and Gregory J. S. Lohman. 2018. Comprehensive profiling of four base overhang ligation fidelity by T4 DNA ligase and application to DNA assembly. *ACS Synthet. Biol.* 7, 11 (Nov. 2018), 2665–2674. DOI: https://doi.org/10.1021/acssynbio.8b00333

[31] Jiayuan Quan and Jingdong Tian. 2009. Circular polymerase extension cloning of complex gene libraries and pathways. *PLoS One* 4, 7 (July 2009), e6441. DOI: https://doi.org/10.1371/journal.pone.0006441

[32] David Reinsel, John Gantz, and John Rydning. 2018. The digitization of the world from edge to core. (2018), 28. IDC White Paper.
[33] Stephen E. Richardson. 1992. *Caching Function Results: Faster Arithmetic by Avoiding Unnecessary Computation.* Technical Report. Sun Microsystems, Inc.
[34] Nathaniel Roquet, HyunJun Park, and Swapnil P. Bhatia. 2020. Nucleic acid-based data storage. Retrieved from https://patents.google.com/patent/US10650312B2/en.
[35] Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. 2004. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol.* 2, 12 (Dec. 2004), e424. DOI: https://doi.org/10.1371/journal.pbio.0020424
[36] Alejandro Sarrion-Perdigones, Erica Elvira Falconi, Sara I. Zandalinas, Paloma Juárez, Asun Fernández-del Carmen, Antonio Granell, and Diego Orzaez. 2011. GoldenBraid: An iterative cloning system for standardized assembly of reusable genetic modules. *PLoS One* 6, 7 (July 2011), e21622. DOI: https://doi.org/10.1371/journal.pone.0021622
[37] Alejandro Sarrion-Perdigones, Marta Vazquez-Vilar, Jorge Palací, Bas Castelijns, Javier Forment, Peio Ziarsolo, José Blanca, Antonio Granell, and Diego Orzaez. 2013. GoldenBraid 2.0: A comprehensive DNA assembly framework for plant synthetic biology. *Plant Physiol.* 162, 3 (July 2013), 1618–1631. DOI: https://doi.org/10.1104/pp.113.217661
[38] Jaswinder Sharma, Rahul Chhabra, Yan Liu, Yonggang Ke, and Hao Yan. 2006. DNA-templated self-assembly of two-dimensional and periodical gold nanoparticle arrays. *Angewandte Chemie Int. Ed.* 45, 5 (2006), 730–735. DOI: https://doi.org/10.1002/anie.200503208
[39] S. M. Hossein Tabatabaei Yazdi, Yongbo Yuan, Jian Ma, Huimin Zhao, and Olgica Milenkovic. 2015. A rewritable, random-access DNA-based storage system. *Sci. Rep.* 5 (Sept. 2015), 14138. DOI: https://doi.org/10.1038/srep14138
[40] Kyle J. Tomek, Kevin Volkel, Alexander Simpson, Austin G. Hass, Elaine W. Indermaur, James M. Tuck, and Albert J. Keung. 2019. Driving the scalability of DNA-based information storage systems. *ACS Synthet. Biol.* 8, 6 (June 2019), 1241–1248. DOI: https://doi.org/10.1021/acssynbio.9b00100
[41] Dan Tulpan, Derek H. Smith, and Roberto Montemanni. 2014. Thermodynamic post-processing versus GC-content pre-processing for DNA codes satisfying the hamming distance and reverse-complement constraints. *IEEE/ACM Trans. Computat. Biol. Bioinf.* 11, 2 (Mar. 2014), 441–452. DOI: https://doi.org/10.1109/TCBB.2014.2299815
[42] G. K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Trans. Consum. Electron.* 38, 1 (Feb. 1992), xviii–xxxiv. DOI: https://doi.org/10.1109/30.125072
[43] Ernst Weber, Carola Engler, Ramona Gruetzner, Stefan Werner, and Sylvestre Marillonnet. 2011. A modular cloning system for standardized assembly of multigene constructs. *PLoS One* 6, 2 (Feb. 2011). Retrieved from https://doi.org/10.1371/journal.pone.0016765
[44] E. Winfree. 1998. *Algorithmic Self-Assembly of DNA.* Ph.D. Dissertation. California Institute of Technology. Retrieved from https://www.dna.caltech.edu/~winfree/old_html/Papers/thesis.pdf.
[45] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. 2016. A comprehensive study of the past, present, and future of data deduplication. *Proc. IEEE* 104, 9 (Sept. 2016), 1681–1710. DOI: https://doi.org/10.1109/JPROC.2016.2571298
[46] Hao Yan, Sung Ha Park, Gleb Finkelstein, John H. Reif, and Thomas H. LaBean. 2003. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science* 301, 5641 (2003), 1882–1884. Retrieved from http://www.jstor.org/stable/3835179.
[47] J. Ziv and A. Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (May 1977), 337–343. DOI: https://doi.org/10.1109/TIT.1977.1055714

Received March 2021; revised November 2021; accepted January 2022